NAVAL POSTGRADUATE SCHOOL
Monterey, California

N35275

THESIS

A FORMAL APPROACH TO HAZARD DECOMPOSITION IN SOFTWARE FAULT TREE ANALYSIS

by

Donald Michael Needham

June, 1990

Thesis Advisor: Timothy Shimeall

Approved for public release; distribution is unlimited.

20100915195

T247296

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release;
distribution is unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Computer Technology Curriculum
Naval Postgraduate School

6b. OFFICE SYMBOL *(if applicable)*
37

7a. NAME OF MONITORING ORGANIZATION
Naval Postgraduate School

6c. ADDRESS *(City, State, and ZIP Code)*
Monterey, CA 93943-5000

7b. ADDRESS *(City, State, and ZIP Code)*
Monterey CA 93943-5000

8a. NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(if applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION N

11. TITLE *(Include Security Classification)*
A FORMAL APPROACH TO HAZARD DECOMPOSITION IN SOFTWARE FAULT TREE ANALYSIS(U)

12. PERSONAL AUTHOR(S)
NEEDHAM, Donald M.

13a. TYPE OF REPORT
Master's Thesis

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Year, Month, Day)*
June 1990

15. PAGE COUNT
75

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of t
Department of Defense or the US Government

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Fault Tree Analysis, Software Safety, Real-time software, Control softwar
Formal Models, Life-critical software, Safety Assessment

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

As digital control systems are used in life-critical applications, assessment of the safety of these control system becomes increasingly important. One means of formally performing this assessment is through fault tree analysi Software Fault Tree Analysis (SFTA) starts with a system-level hazard that must be decomposed in a largely-huma intensive manner until specific module of the software system are indicated. These modules can then be formall analyzed using statement templates.

The focus of this thesis is to approach the decomposition of a system-level hazard from a formalized standpoin Decomposition primarily proceeds along two distinct but interdependent dimensions, specificity of event an subsystem size. The Specificity-of-Event dimension breaks abstract or combined events into the specific syste events that must be analyzed by the fault tree. The Subsystem-Size dimension deals with the scope of the hazard, an itemizes the subsystems where localized events may lead to the hazard. Decomposition templates are developed i this thesis to provide a framework for decomposing a system-level hazard to the point at which line-by-line cod analysis can be conducted with the existing statement templates. These templates serve as guides for conducting th decomposition, and ensure that as many as possible of all the applicable decomposition aspects are evaluated.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Timothy J. Shimeall

22b. TELEPHONE *(Include Area Code)*
(408) 646-2509

22c. OFFICE SYMBOL
CS/Sm(52Sm)

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

A Formal Approach to
Hazard Decomposition
in
Software Fault Tree Analysis

by

Donald Michael Needham
Lieutenant, United States Navy
B.S., United States Naval Academy, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1990

Author: Donald Michael Need
Donald Michael Needham

Approved by: Timothy Shimeall
Timothy Shimeall, Thesis Advisor

LCDR Leigh Bradbury, Second Reader

Robert B. McGhee
Robert B. McGhee, Chairman
Department of Computer Science

ABSTRACT

As digital control systems are used in life-critical applications, assessment of the safety of these control systems becomes increasingly important. One means of formally performing this assessment is through fault tree analysis. Software Fault Tree Analysis (SFTA) starts with a system-level hazard that must be decomposed in a largely-human-intensive manner until specific modules of the software system are indicated. These modules can then be formally analyzed using statement templates.

The focus of this thesis is to approach the decomposition of a system-level hazard from a formalized standpoint. Decomposition primarily proceeds along two distinct but interdependent dimensions, specificity of event and subsystem size. The Specificity-of-Event dimension breaks abstract or combined events into the specific system events that must be analyzed by the fault tree. The Subsystem-Size dimension deals with the scope of the hazard, and itemizes the subsystems where localized events may lead to the hazard. Decomposition templates are developed in this thesis to provide a framework for decomposing a system-level hazard to the point at which line-by-line code analysis can be conducted with existing statement templates. These templates serve as guides for conducting the decomposition, and ensure that as many as possible of all the applicable decomposition aspects are evaluated.

Thesis
N35275
C.1

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 1

1. Hardware Fault Tree Analysis 1

a. Logical Relationship Between Hardware Events 1

b. Orders of Fault Events 3

2. Software Fault Tree Analysis 6

a. Statement Templates 8

b. Contradiction Within Fault Trees 9

B. SAFETY CRITICAL CONSIDERATIONS 10

C. PROBLEM STATEMENT 11

D. STRUCTURE OF THESIS 13

II. DECOMPOSITION TEMPLATES 14

A. DECOMPOSITION OF A HAZARD 14

B. SOFTWARE SYSTEM REQUIREMENTS 15

C. SPECIFICITY-OF-EVENT DECOMPOSITION DIMENSION 16

D. SPECIFICITY-OF-EVENT DECOMPOSITION TEMPLATES 16

1. OR Template 16

2. NOT Template 18

3. AND Templates 19

E. HIGH-LEVEL TRAFFIC-LIGHT DESIGN 22

F. SUBSYSTEM-SIZE DECOMPOSITION DIMENSION 24

DUDLEY KNOX LIBRARY

G. SUBSYSTEM-SIZE DECOMPOSITION TEMPLATES 25

  1. Subsystem Template 25

  2. Process Template 27

  3. Communication Template 30

  4. Access Template 33

H. TIME RELATIONSHIP OF EVENTS 36

I. INTERDEPENDENCE OF DECOMPOSITION DIMENSIONS 39

III. APPLICATION OF DECOMPOSITION TEMPLATES 40

A. ONE-LANE BRIDGE REQUIREMENTS MODEL 40

B. ONE-LANE BRIDGE HAZARD 41

C. DECOMPOSITION OF ONE-LANE BRIDGE HAZARD 41

  1. Specificity-of-Event Decomposition Dimension 41

  2. Subsystem-Size Decomposition Dimension 44

  3. Overall Fault Tree for Bridge Hazard 51

D. SUMMARY OF DECOMPOSITION TEMPLATE APPLICATION . . . 51

IV. CONCLUSIONS AND RECOMMENDATIONS 54

A. CONCLUSIONS 54

  1. Relevance to MIL-STD-882B 54

  2. Advantages of Decomposition Templates 55

  3. Limitations of Decomposition Templates 56

B. RECOMMENDATIONS FOR FURTHER RESEARCH 57

APPENDIX A. REQUIREMENTS MODEL RULES FOR ONE-LANE BRIDGE . 58

APPENDIX B. GRAPHICAL REPRESENTATION OF ONE-LANE BRIDGE . . . 59

LIST OF REFERENCES 62

INITIAL DISTRIBUTION LIST 63

LIST OF FIGURES

Figure 1. Relevant Fault Tree Symbols 2

Figure 2. Logical OR and Logical AND 3

Figure 3. Simple Electrical Circuit 4

Figure 4. Ada Assignment Statement Template 9

Figure 5. Decomposition Dimensions 14

Figure 6. OR Template 17

Figure 7. Example OR Template Use 18

Figure 8. NOT Template 19

Figure 9. Example NOT Template 19

Figure 10. AND Template 20

Figure 11. Example AND Template Use 21

Figure 12. AND (WHILE/WHEN) Template 21

Figure 13. Example AND (WHILE/WHEN) Template Use 22

Figure 14. High-Level Traffic-Light Design 23

Figure 15. Subsystem Template 25

Figure 16. Example Subsystem Template Use 26

Figure 17. Process Template 27

Figure 18. Example Process Template Use 29

Figure 19. Communication Template 30

Figure 20. Example Communication Template Use 32

Figure 21. Access Template 33

Figure 22. Access To Device Template 34

Figure 23. Access To Database Template 35

Figure 24. Example Access To Device Template Use 36

Figure 25. Time Template 37

Figure 26. Example Time Template Use 38

Figure 27. Depiction of One-Lane Bridge 40

Figure 28. Bridge Hazard Decomposition 42

Figure 29. Decomposition of Bridge Hazard Event 43

Figure 30. Process Subsystem Decomposition 46

Figure 31. Process Template 47

Figure 32. Access Subsystem Decomposition 47

Figure 33. Access Template 48

Figure 34. Communication Subsystem Decomposition 49

Figure 35. Communication Template 50

Figure 36. Overall Fault Tree for Bridge Hazard 52

ACKNOWLEDGEMENTS

I would like to thank Prof. Timothy Shimeall and Lcdr. Leigh Bradbury for taking the time to support, encourage and guide me. I would like to thank David L. Ripps and Prentice Hall, Inc., for permission to reprint the material from AN IMPLEMENTATION GUIDE TO REAL-TIME PROGRAMMING, 1990, pp. 35-39, that appears in appendices A and B and is quoted in Chapter III. Finally, I would like to thank my wife Catherine and daughter Mary for their love, patience and understanding.

I. INTRODUCTION

A. BACKGROUND

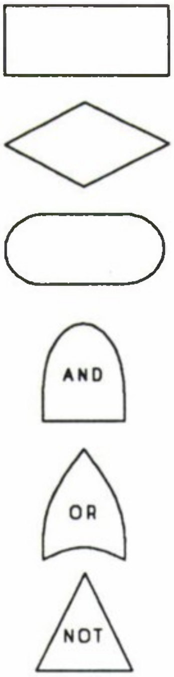
A fault tree is a graphical representation of an analysis of a physical system. This analysis shows whether a combination of one or more contributing causes can result in the occurrence of a specific undesired event. The starting point for the construction of a fault tree is the specification of an undesired event. Fault tree analysis has been applied to both the hardware and software of systems as a means of proving an undesired event. The principles that are used to analyze software and hardware via fault trees are sufficiently different to consider hardware and software fault tree analysis as two separate processes.

1. Hardware Fault Tree Analysis

Fault tree analysis was first applied to hardware systems, and made extensive use of hardware component failure rates. With hardware, the probability of a specific piece of hardware failing due to wear can be accurately determined. Either the analysis of historical data or direct experimentation can be used to assign a probability of failure to virtually any piece of hardware. Logic theory in combination with probability of failure data forms the mathematical basis for fault tree analysis.

a. Logical Relationship Between Hardware Events

The symbols used to build the fault trees presented throughout this thesis are shown in Figure 1. These symbols are a subset of the symbols presented by Hammer [Ref. 1:pp. 227-229] that are commonly used in both hardware and software fault tree analysis. Bagchi [Ref. 2:p. 4486] shows how two basic logic properties, OR and AND can be used to relate the probabilities of independent events. Consider, for example, the following two equations where A, B, C are independent events and P(A),



The RECTANGLE indicates an event to be analyzed further

The DIAMOND is used for non-primal events not developed further for lack of information or insufficient consequences

The OVAL is used to indicate a condition

The AND gate indicates that all input events are required in order to cause the output event

The OR gate indicates that one or more of the input events are required in order to cause the output event

The NOT gate indicates that the output event is the logical negation of the input event

Figure 1. Relevant Fault Tree Symbols

P(B), P(C) are equal to the probability of the occurrence of the events A, B, C respectively. For the purposes of demonstrating the logical AND and OR properties, let Equation (1) be, "if C=A+B then P(C)=P(A)+P(B)-P(A^B)", and let Equation (2) be, "if C=A*B then P(C)=P(A)*P(B)". In both Equation (1) and Equation (2), P(X) is the probability of independent event X occurring. Equation (1) can be reduced to P(C)=P(A)+P(B), since the purpose of subtracting the probability of the intersection of two events is to remove the redundancy involved when two OR'd events occur simultaneously. With respect to fault tree analysis, this subtraction is not strictly necessary. If two OR'd events occur simultaneously, the resultant event holds just as if either one of the OR'd events had occurred separately, and in many systems the probability of two simultaneous component failures is sufficiently low as to be negligible.

Equations (1) and (2) form the basic foundation of the graphical depiction of a fault tree, although any logical relation (XOR, NAND, NOR, etc.) can be used as required to correctly represent the system. Bagchi represents the logical OR of Equation (1) and the logical AND of Equation (2) graphically in Figure 2.

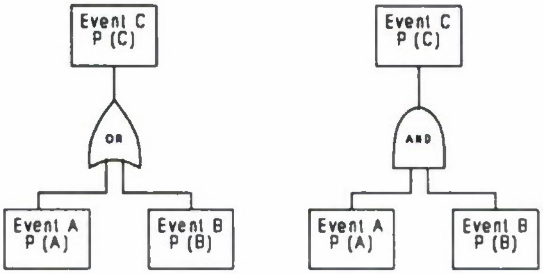


Figure 2. Logical OR and Logical AND

The OR gate of Figure 2 shows the resultant fault tree from a system that has two components in series. A failure causing either event A or event B will result in the event C. The AND gate of Figure 2 shows the fault tree from a system that has two parallel events. As indicated by the AND gate, both events A and B must occur in order for event C to result. [Ref. 2:p. 4486] In short, fault tree analysis can be used to identify combinations of events (hardware failure, design flaws) that have the potential of causing the occurrence of the specified undesired event.

b. Orders of Fault Events

An important aspect of the use of fault tree analysis is the decomposition of an undesired event into sub-events. These sub-events must be further decomposed until system components or actions are identified. If these system events can be proven to be incapable of occurring, the system is proven incapable of generating the undesired event.

Fussell developed a methodology called the Synthetic Tree Model (STM) [Ref. 3:p. 425] that distinguishes between four orders of fault events to guide in the decomposition of an undesired event. The orders represent various degrees of abstraction from the actual system components. The lowest order event, first order, is the most abstract. The undesired event that the fault tree is attempting to prove or disprove is the first order event that serves as the starting point or "root" of the analysis tree.

Consider as an example the simple electrical circuit in Figure 3, similar to the circuit presented by Salem [Ref. 4:p. 54] in his work on fault tree construction. For the purpose of presenting Fussell's orders of fault events, this example circuit is

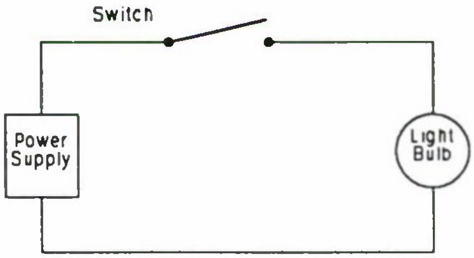


Figure 3. Simple Electrical Circuit

composed of a light bulb, a power supply and a switch for turning the light on and off. The undesired event that might be used as a first order event could be the event "light not on". With Fussell's methodology, the fault tree is constructed by manually decomposing the first order event into higher order (less abstract) events. The undesired event "light not on" could be broken down into the less abstract events "no power to light bulb" or "light bulb broken".

Second order fault events are conditions that effect a grouping of components that if failed would result in the occurrence of the first order event. A second order event for our example circuit could be "no power in group A", where group A consists of all components effected by this second order event. In our example, group A would be comprised of the power supply, light bulb and light switch. These components are grouped with respect to the serial nature of electronic circuits, in that the failure of any one component would necessarily result in the apparent failure of all the other components in the group.

A third order event involves a fully functional component that merely acts failed due to a failure in another part of the system. The "no power to light bulb" event decomposed from the original undesired event is an example of a third order event. The light bulb will act failed (not light up) when in fact the light bulb is functional but has no power coming into it. To develop this third order event, STM requires a search for all second order groups in which the light bulb appears. The "no power to light bulb" event requires "no power" in each of these second order groups involving light bulbs. In other words, these groups are AND'd together to yield the third order event "no power in light bulb". Thus, a third order event is comprised of a combination of applicable second order events.

The final level in Fussell's methodology is the fourth order event. A fourth order event involves a component that acts failed due to the input that the component receives. The light switch being in the off position would be an example of an input that causes the light bulb to act as though it had failed.

The development of Fussell's methodology was based on electronic circuit principles. Second order events make use of the serial path of an electrical current. This reliance on the serial nature of components in an electrical system is a critical aspect of Fussell's STM methodology. It is this serial nature that requires the grouping of all the components that will also fail if any component along the electrical path fails. This serial grouping forms the basis for linking a third order event by an AND of a group of second order events. This serial grouping allows Fussell's methodology to require that all groups containing a component failure that would yield the third order event be included in the AND.

Fussell acknowledges that the fault trees constructed using his Synthetic Tree Model are quite lengthy. The length of his trees is attributable to the method by which Fussell's methodology groups second order events. Second order events require the grouping of all components serially connected to a component whose failure would result in the second order event. This lengthens the tree by adding on components that a fault tree developed using Fussell's methodology must check. A tree being developed by a method other than Fussell's might not require this type of grouping, and might therefore have less branches to check. There is, however, an advantage attributable to the length of Fussell's trees. Any number of fault trees constructed independently for the same system and main failure event using Fussell's methodology will result in identical fault trees [Ref. 3:p 432]. This indicates a level of reliability in Fussell's methodology, and suggests the possibility that part or all of the fault tree construction may be automated.

2. Software Fault Tree Analysis

Leveson and Harvey [Ref. 5:p. 570] describe an analysis technique that makes use of the basic principles of fault tree analysis, but applies them to the analysis of the software of a system rather than to a system's hardware components. Since the basic notion of determining what sequence of events are capable of producing the undesired event is the same for the analysis of both hardware and software, a software tree can be linked together with a hardware tree at the appropriate interface. This allows the analysis of an undesired event to span over the entire hardware/software system. This linkage ability strengthens software fault tree analysis (SFTA) by allowing the effects of a hardware failure on the software system to be analyzed.

The value of being able to link together both the hardware and the software of a system when conducting software fault tree analysis is illustrated by an example of a hazard resulting from missile launch control software being tested for the F/A-18 aircraft [Ref. 6:p. 3]. The computer system was being used to control the launch of a wing mounted missile. The computer system was to fire the missile, open the wing station clamp to release the missile, and then close the clamp. The hazard occurred because the software closed the clamp before the missile had built up enough thrust to leave the wing. This resulted in the aircraft having 3000 pounds of extra thrust attached to the wing of the aircraft. The ability of SFTA to be linked in with a hardware fault tree presents the opportunity to fully analysis these interrelated errors.

There are differences in the approach that software fault tree analysis takes in the analysis of a software system that sets this method apart from the more traditional hardware fault tree analysis methods. A hardware fault tree relies on the probabilities of failure for various hardware components. These failure probabilities are either known, or can be accurately predicted. Hardware fault tree analysis makes use of these probabilities of failure to construct a tree that will show which components are critical to the system. This tree construction based on failure probabilities is possible since hardware components are assumed to fail independently at a determinable rate. Software fault tree analysis cannot proceed in the same manner since no such assumption about failure independence can be made about software component failures.

[Ref. 5:p. 576] Furthermore, since software faults are design faults, determination of a software failure rate is an unsolved problem [Ref. 7:p. 104].

The approach taken by Leveson and Harvey's software fault tree analysis is to use the fault tree to show that the logic of the software design will not produce or contribute to actions that lead to system failures. This analysis can take place at any desired level of abstraction, including code level analysis. If a failure is shown to be possible, the structure of the resultant tree determines any environmental conditions that could lead to the software causing a safety failure. [Ref. 5:p. 576] Software fault tree analysis starts with the assumption that the software system has gotten itself into a state in which the undesired event has occurred. With this assumption, the fault tree is developed backwards starting with the undesired event and examines all the possible paths that lead to the undesired event. Typical usage starts with the program statements as written in the implementation language. Each statement is assumed to have executed in a manner such that the undesired event occurs. The development proceeds by looking at each statement and determining how the assumed statement execution was allowed to occur.

a. Statement Templates

Cha, Leveson and Shimeall [Ref. 8:pp. 380-383] present statement templates (hence forth referred to as "Cha's Templates" for convenience) that are used to analyze each of the possible Ada statements. The template for the decision statement shows all the pathways that the software can take when a decision statement is executed. The template for non-decision statements can be used to analyze the various possible results from each statement, and can show whether the execution of the assignment statement could result in another link in the chain leading back to the undesired event. Figure 4 is an example of a template developed for an Ada assignment statement [Ref. 8:p. 380].

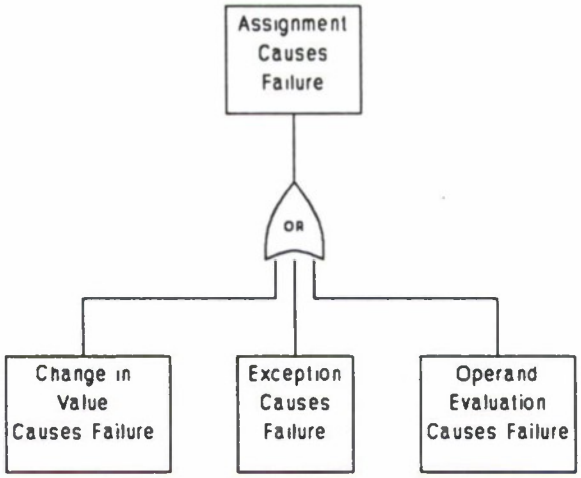

Assignment Causes Failure
OR
Change in Value Causes Failure
Exception Causes Failure
Operand Evaluation Causes Failure

Figure 4. Ada Assignment Statement Template

Cha's templates were designed using statement semantics from the Ada language, and by analyzing the causes of frequently made programming errors. The templates suggest possible branches to analyze specific statements. This aids the analyst in the consideration of all the possible results of a statement. The authors contend that templates could be constructed to analyze program statements written in any language, and allude that the templates might be tailor made to try to uncover specific errors. This tailoring might make the search for critical, system specific events more effective.

b. Contradiction Within Fault Trees

With or without the guidance of statement templates, every single safety-critical path that the software may take during the course of execution must be analyzed via a branch in the fault tree. Ideally, as paths are analyzed, contradictions will occur. A contradiction with regard to software fault tree analysis is when an event that could cause the undesired event is prevented by the software. Since the contradicted event cannot occur, that particular analysis branch need no longer be pursued.

A path that terminates without a contradiction is a path via which the undesired event can occur, and actually proves that the undesired event could possibly be generated by the system software. Since the exact path that allows the undesired event is now known, it is a simple matter to place run-time checks in the code to halt the pathway to the undesired event.

Even with the ability to halt tree development after a contradiction, any given fault tree may still be lengthy if carried out to full term. Leveson and Harvey [Ref. 4:p. 576] point out that it is not necessary to expand the entire tree down until every path terminates in either a contradiction or the system environment. At any level of abstraction, the analysis may be halted and run-time checks inserted into the code to trap a developing unsafe state.

B. SAFETY CRITICAL CONSIDERATIONS

The prospect of testing a system using fault tree analysis may seem tempting since this analysis method effectively proves whether or not a system will allow a specific event to occur. However, testing deals primarily with determining whether or not a system will function as it was designed to. The purpose of testing a system is to attempt to show that the system does not meet its specification [Ref. 9:p. 3].

Leveson and Harvey [Ref. 5:p. 571] point out that software fault tree analysis is focused on proving that the system is incapable of producing a specific event from a software standpoint. This is much different from testing to see if a system will provide correct output for all of a potentially infinite set of states that the system could be in during the course of execution. Focusing on proving that the system cannot produce a specific undesired event allows the analysis to take advantage of path termination when a contradiction occurs, which makes for shallower trees. Still, the set of what a software system is not supposed to do is the complement of what the system is supposed to do, with the size of both sets potentially being very large. To attempt to use software fault tree analysis on the set of events that the system is not supposed to do would require an impossibly large number of fault trees to be developed. Since the development of a fault tree is generally not a trivial undertaking, the set of events to be analyzed must be reduced. The reduction of this set is where safety criticality comes in.

The inclusion of safety criticality as a means of reducing the set of undesired events with which the software fault tree analysis must contend primarily relies on two underlying concepts. As Cha [Ref. 8:p. 377] points out, the first concept is that not all failures are of equal consequence, and the second concept is that the number of potentially serious failures is relatively small as compared to the overall set of undesired events. Reducing the set of undesired events by determining which events are critical to the safe operation of the software system allows SFTA to be focused on the smaller set of potentially high-cost or otherwise unacceptable errors.

C. PROBLEM STATEMENT

The templates presented by Cha are useful in guiding the statement by statement decomposition of software fault trees. However, these templates are limited in application to line by line code analysis, and do not deal with undesired events not linked to single linear sequences of statements. With current software fault tree analysis techniques, no formal method exists for decomposing a system-level hazard to the point at which the Cha statement templates can be applied. System-level hazards are currently decomposed manually in what proves to be a largely human intensive manner. Decomposition of system-level hazards in this manner relies heavily on human insight and knowledge of the software system, and introduces human error in the form of oversight.

This thesis approaches the decomposition of a system-level hazard from a formalized standpoint. The decomposition of the system-level hazard primarily proceeds along two distinct but interdependent dimensions, specificity of event and subsystem size. The hazard starts as a more general event that is assumed to occur somewhere within a large system or subsystem. The process of decomposition involves breaking the hazard into more specific events, or itemizing subsystems within which the hazard may be a consequence. Decomposing a hazard in one dimension often results in opening up further possibilities for decomposition along the other dimension.

The Specificity-of-Event decomposition dimension deals with how specific the events and conditions associated with the hazard are. The goal of this decomposition dimension is to identify the particular events that are to be analyzed within the subsystems that comprise the system being examined. This decomposition works by itemizing each event that could cause the hazard in the current subsystem. For this itemization to be effective, the scope of the decomposition must constitute a complete enumeration of the ways in which the hazard could occur within the current subsystem. Although some of these itemized events will not be considered further, due to the contradictions that the events represent, all possible occurrences of the hazard are itemized for completeness. The sum of these itemized events represents in total the ways in which the hazard can occur within the scope of the current subsystem.

The Subsystem-Size decomposition dimension deals with how large of a subsystem is being considered when determining how the hazard being analyzed can occur. This decomposition dimension moves the scope of the hazard from a more general system wide event, to an event that is more specific in terms of the scope of the subsystem that the event effects. This increase in the specificity of the event effectively works to identify the specific system events that must be targeted for analysis by the statement templates. Decomposing the hazard into various subsystems focuses the tree towards subsystems that play a role in the hazard being analyzed.

D. STRUCTURE OF THESIS

Chapter II develops the design of templates that provide a framework for decomposing a system-level hazard to the point at which line by line code analysis can be conducted with existing statement templates. These templates serve as guides for conducting the decomposition, and ensure that as many as possible of all the applicable decomposition aspects are evaluated.

Chapter III demonstrates the use of the decomposition templates through an example. The fault tree is developed starting with the decomposition of an undesired system-level event. Events are further decomposed using the templates developed in Chapter II indicating specific modules of the software system along with associated specific events and conditions. This decomposition process is continued until the point at which Cha's statement templates can be implemented.

Chapter IV summarizes the work accomplished through the use of the decomposition templates, and the conclusions that were drawn through the development of this thesis. Areas of future work in the area of software event decomposition templates are indicated in this chapter.

II. DECOMPOSITION TEMPLATES

A. DECOMPOSITION OF A HAZARD

The hazard at the root of the software fault tree is decomposed in a manner such that statement templates may be used to analyze the hazard at the statement level of each applicable subsystem. The hazard is decomposed in two different abstract dimensions. As depicted by Figure 5, these decomposition dimensions are the specificity of the event being considered, and the size of the subsystem being considered.

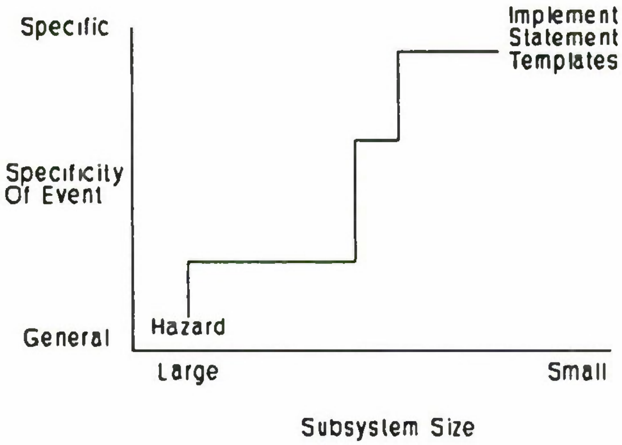

Specific
Specificity Of Event
General
Hazard
Implement Statement Templates
Large
Small
Subsystem Size

Figure 5. Decomposition Dimensions

Figure 5 shows the hazard starting as a more general event that occurs somewhere in a large system or subsystem. The process of decomposition involves breaking the hazard into more specific events, or itemizing subsystems within which the hazard may be a consequence, or both. In both dimensions of the graph in Figure 5, the path that the decomposition process takes is non-decreasing in nature. For example, the Specificity-of-Event dimension may be held constant so as to allow the other dimension, Subsystem-Size, to move more to the right as it is decomposed. However, the Specificity-of-Event dimension does not move towards a more general event to allow the Subsystem-Size dimension to move from a larger to a smaller subsystem size. A move such as this results in backtracking within the analysis tree. Although a backtracking move would be required when the analyst determines that a previous move was incorrect, backtracking effectively accomplishes nothing more than returning to a previous state so that the analysis can take an alternate avenue.

B. SOFTWARE SYSTEM REQUIREMENTS

In order to illustrate each decomposition template as it is presented, a simple software program is used. The following requirements for an imaginary traffic light control program for an automobile intersection was used by Cha, Leveson and Shimeall in demonstrating the use of software statement templates, and is repeated here :

> "A traffic light control system at an intersection consists of four (identical) sensors and a central controller. The sensors in each direction detect cars approaching the intersection. If the traffic light currently is not green, the sensor notifies the controller so that the light will be changed. A car is expected to stop and wait for a green light. If the light is green already, the car may pass the intersection without stopping. The controller accepts change requests from the four sensors and arbitrates the traffic light changes. Once the controller changes the light in one direction (east-west or south-north) to green, it maintains the green signal for five seconds so that other cars in the same direction may pass in the same direction without stopping. Before the green light in any direction becomes red, it should remain in yellow for one second so that any car present in the intersection may clear. The light then turns to red while the light in the opposite direction turns to green." [Ref. 8:p. 382]

With the above requirements, the hazard analyzed to illustrate the decomposition templates is "Two Cars Simultaneously in Intersection Traveling in Perpendicular Directions", or "Perpendicular Cars" for short. As noted by Cha, the above requirements were designed to contain several failure modes. Finding failure modes other than the one illustrated by Cha have no relevance to the importance of Cha's work. Hazard decomposition as presented herein is intended to precede and complement Cha's work on statement templates. The application of statement templates completes the hazard decomposition presented in this thesis.

C. SPECIFICITY-OF-EVENT DECOMPOSITION DIMENSION

The Specificity-of-Event decomposition deals with how specific the events and conditions associated with the hazard are. The goal of decomposing the hazard with regard to specificity is to identify the particular events that are to be analyzed within the subsystems that comprise the system being examined. This decomposition works by itemizing each event that could cause the hazard in the current subsystem. For this itemization to be effective, the scope of the decomposition must constitute a complete enumeration of the ways in which the hazard could occur within the current subsystem. Although some of these itemized events will not be considered further due to the contradictions that the events represent, all possible occurrences of the hazard are itemized for completeness. The sum of these itemized events represents in total the ways in which the hazard can occur within the scope of the current subsystem.

D. SPECIFICITY-OF-EVENT DECOMPOSITION TEMPLATES

Three basic logic relations, OR NOT and AND, are used to determine the relationship of the events itemized by the decomposition process. The following templates show the logical relationship that the decomposed event(s) and condition(s) have to the hazard.

1. OR Template

The OR Template (Figure 6) is used to decompose a hazard that occurs because either one of two events (Event 1 or Event 2) occur. Both Event 1 and Event 2 are capable of singularly causing the hazard to occur. Since either Event 1 or Event 2 could cause the hazard, each event is given its own branch within the OR gate. The

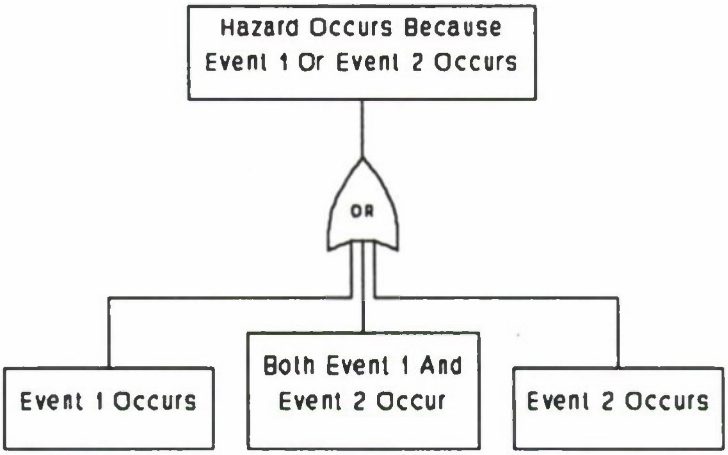

Hazard Occurs Because Event 1 Or Event 2 Occurs
OR
Event 1 Occurs
Both Event 1 And Event 2 Occur
Event 2 Occurs

Figure 6. OR Template

center branch within the OR gate is reserved for the instance in which both Event 1 and Event 2 occur simultaneously. Further decomposition on this center branch generally is halted immediately because in order for this branch to be taken during analysis, both events must occur. If both events occur, then the hazard will occur the same as if only one of the events had occurred. The analysis of the center branch may be a direct duplication of the analysis of the two separate events, and in that case it is not further pursued. Only if the center event suggests important and non-duplicative analysis directions is it explored. For example, if the two events leading to a hazard are not independent, the center branch could be used to explore their interdependency as a source of the undesired event. If the center branch is irrelevant, it is omitted from the analysis.

An example of the decomposition of the hazard using a OR Template (Figure 7) in the traffic light control system is found in the requirements. The requirements state that before the green light in any direction becomes red, it should

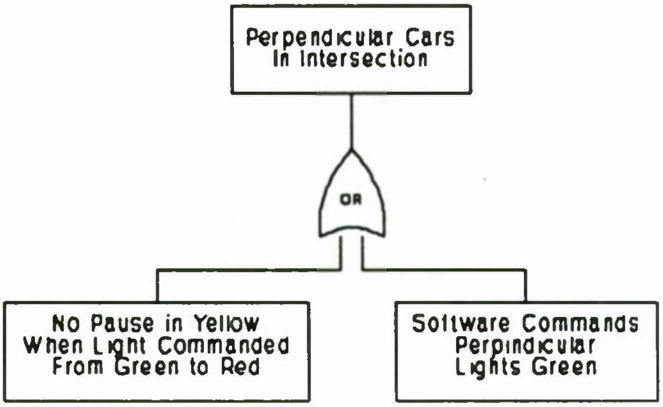

Perpendicular Cars In Intersection
OR
No Pause in Yellow When Light Commanded From Green to Red
Software Commands Perpindicular Lights Green

Figure 7. Example OR Template Use

remain in yellow for one second so that any car presently in the intersection may clear. If the software fails to make each light pause in yellow for one second during the transition from green to red, a car could get caught in the intersection when the opposing traffic light turns green. One branch of the OR Template for the decomposition of Perpendicular Cars is the event "No Pause in Yellow When Light Commanded From Green to Red". Another branch considered here is the software system's command of the lights. If the software were to command perpendicular lights to be green at the same time, the Perpendicular Cars hazard could occur. This decomposition represents the second branch as "Software Commands Perpendicular Lights Green".

2. NOT Template

The NOT Template (Figure 8) is used to decompose a hazard that occurs because an event or action fails to occur. The logical NOT decomposes the hazard such that it is not the case that the event does occur. This frees the analyst to consider the case in which the event does occur. As with a mathematical proof by contradiction, it may be more straightforward to show that there is a contradiction with assuming that an event does occur than it is to show that, for all cases, it is impossible for the event to occur within the system. An example of the use of a NOT Template (Figure 9) is the "No Pause in Yellow When Light Commanded From Green to Red" Branch of the example OR Template hazard decomposition appearing in Figure 7. By applying the NOT Template, the event "Light Pauses in Yellow When Light Goes From Green to Red" could instead be analyzed if this event is deemed more straightforward to evaluate.

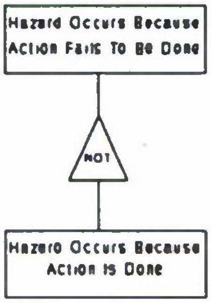

Hazard Occurs Because Action Fails To Be Done
NOT
Hazard Occurs Because Action Is Done

Figure 8. NOT Template

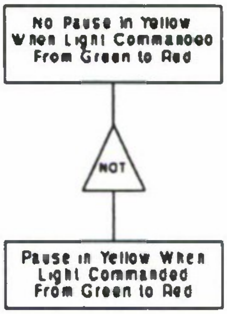

No Pause in Yellow When Light Commanded From Green to Red
NOT
Pause in Yellow When Light Commanded From Green to Red

Figure 9. Example NOT Template

3. AND Templates

A logical AND is represented with two different AND decomposition templates. The first AND Template (Figure 10) is used to decompose a hazard that occurs because both of two separate events (Event 1 and Event 2) occur. This

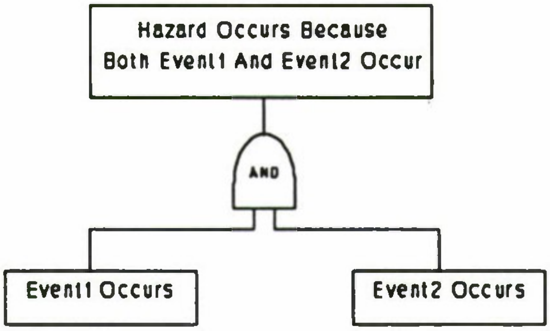


Figure 10. AND Template

decomposition template breaks the hazard into two events, each of which is analyzed individually. If either of the events can be shown to be incapable of occurring, the analysis of both of the AND branches is stopped.

An example of the use of the AND Template (Figure 11) is to further decompose the "Software Commands Perpendicular Lights Green" branch of the OR Template. To decompose this branch further, consideration must be given as to how the software could end up in a state in which perpendicular lights have been commanded green. One way for this event to occur is if the software properly commands a light green, while at the same time the perpendicular light is stuck in a green state.

The second AND (WHILE/WHEN) Template (Figure 12) is used to decompose a hazard that occurs because an event occurs while a specific condition is true. If the condition does not hold while the event occurs, then the hazard does not occur. Thus, the event is dependant upon the condition associated to it by the AND

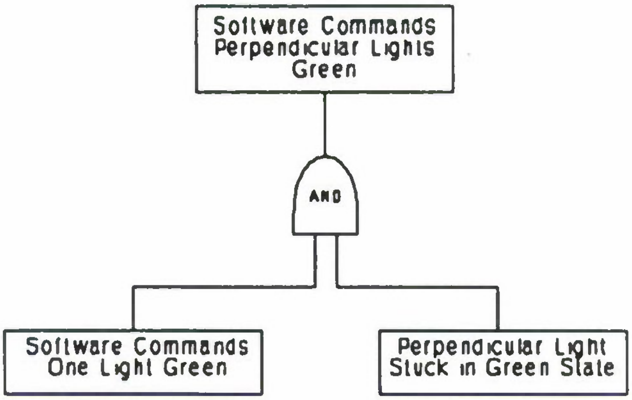

Software Commands Perpendicular Lights Green
AND
Software Commands One Light Green
Perpendicular Light Stuck in Green State

Figure 11. Example AND Template Use

gate. All subsequent decompositions resulting from the event side of the AND gate carry with them the scope of the condition side.

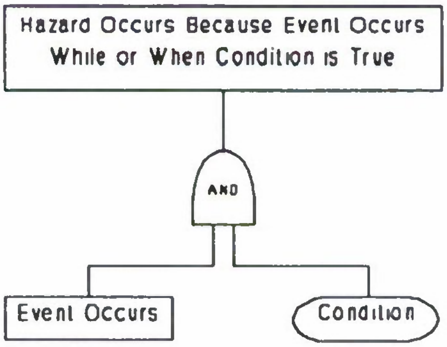

Hazard Occurs Because Event Occurs While or When Condition is True
AND
Event Occurs
Condition

Figure 12. AND (WHILE/WHEN) Template

An example of the use of the AND (WHILE/WHEN) Template (Figure 13) is to further decompose the "No Pause in Yellow When Light Commanded From Green to Red" branch of the example OR Template hazard decomposition in Figure 7. This event is broken down into an event and a condition that must be true in order for the associated event to result in the occurrence of the hazard. The event is decomposed using an AND (WHILE/WHEN) Template into the event "Light Does Not Pause in Yellow" and the condition "Light Commanded From Green to Red".

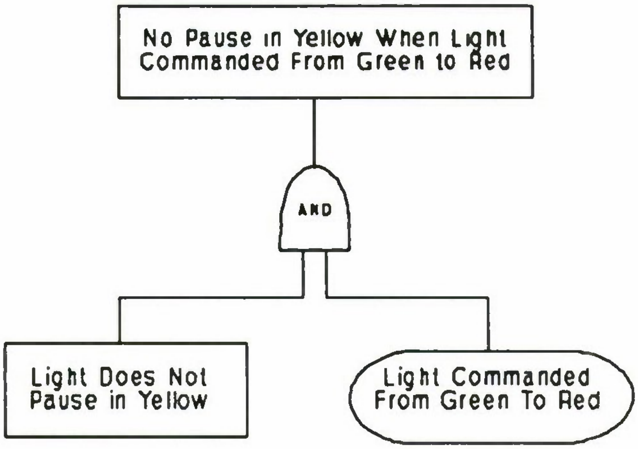

No Pause in Yellow When Light Commanded From Green to Red
AND
Light Does Not Pause in Yellow
Light Commanded From Green To Red

Figure 13. Example AND (WHILE/WHEN) Template Use

E. HIGH-LEVEL TRAFFIC-LIGHT DESIGN

For the purposes of demonstrating the subsystem decomposition templates, a simple high level design loosely fulfilling the requirements for Cha's stoplight system is represented in the High-Level Traffic-Light Design (Figure 14). This high-level design has a "controller" process which determines how the four lights should be set, and sends commands for changing the lights to four identical "Set Signal" processes. The four "Set Signal" processes in turn control the four traffic light devices.

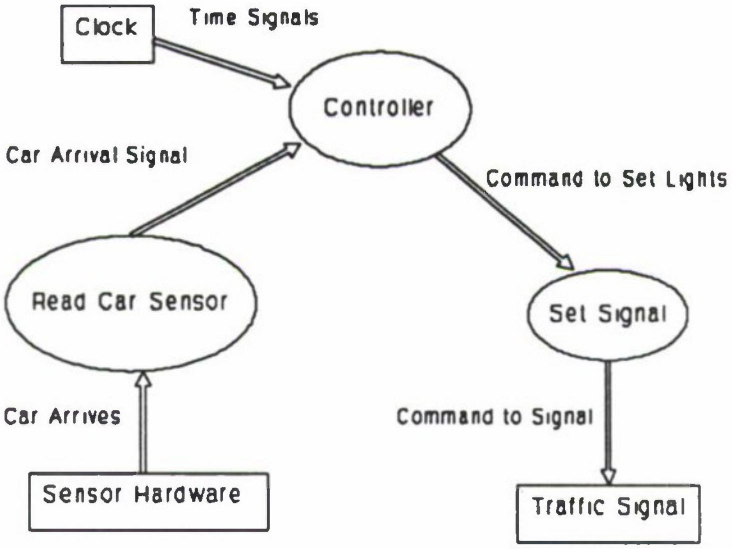

Clock
Time Signals
Controller
Car Arrival Signal
Command to Set Lights
Read Car Sensor
Set Signal
Car Arrives
Command to Signal
Sensor Hardware
Traffic Signal

Figure 14. High-Level Traffic-Light Design

When the "Controller" process determines a light should be changed from green to red, the "Controller" process is responsible for ensuring that the light pauses in yellow during the transition. Each "Set Signal" process controls one of the four physical traffic signals in the intersection, and sends green, yellow, and red indications to the traffic light as directed by the "Controller" process. The "Controller" process determines how the four lights should be set by accepting input both from a clock and from the four identical "Read Car Sensor" processes. Each "Read Car Sensor" process reads sensor hardware that indicates when a car has approached the intersection. The "Controller" process uses input from the clock device to ensure that the light has remained green in one direction long enough to allow a predetermined minimum number of cars through the intersection. This high-level design is by no means complete, and is intended to be a tool for demonstration of the use of subsystem decomposition templates.

F. SUBSYSTEM-SIZE DECOMPOSITION DIMENSION

The Subsystem-Size decomposition dimension deals with how large a subsystem is being considered when determining how the hazard being analyzed can occur. The size of the subsystem may start out comprising the entire software and hardware system in the beginning of the decomposition. This decomposition moves from a more general system wide event, to an event that is more specific in terms of the scope of the subsystem that the event effects. This increase in the specificity of the event effectively works to identify the specific system events that must be targeted for analysis by the statement templates. Decomposing the hazard into various subsystems focuses the tree towards subsystems that play a role in the hazard being analyzed.

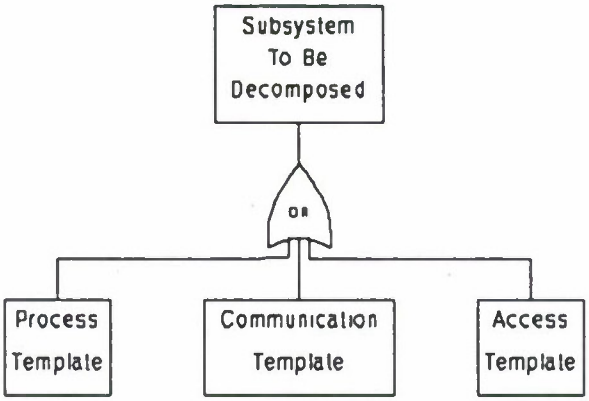
Along with indicating which subsystems should be investigated, the software fault tree must indicate any conditions or events that must be considered when analyzing the subsystem on a statement by statement basis. These conditions and events are indicated through the previously described AND (WHILE/WHEN) template (Figure 12). The focus of the templates presented below is to decompose the overall software system with the intent of identifying the subsystems that can contribute to the occurrence of the hazard.The decomposition of a system into its various subsystems relies on the communication links and interfaces with which the system will function. In order to demonstrate the use of subsystem decomposition templates, the high-level design for the traffic light system previously described is used to represent the communication between the processes of the subsystem.

G. SUBSYSTEM-SIZE DECOMPOSITION TEMPLATES

1. Subsystem Template

Subsystem decomposition templates are used to break out from the overall system the various systems and subsystems that must be considered in regard to the hazard. Three templates are used for the decomposition of subsystems, and are related via an OR gate as shown in the Subsystem Template (Figure 15).

At the root of the Subsystem Template is the subsystem that is indicated by the hazard or event. The Process Template branch decomposes the subsystem based upon the input-process-output of a subsystem. The Communication Template branch decomposes the subsystem based upon the communication between various processes within a subsystem. The Access Template branch decomposes the subsystem based upon the access that a subsystem may have to a device or database.



INPUT/PROCESS/OUTPUT

COMMO B/T PROCESS

WHAT SUB CAN ACCESS

Figure 15. Subsystem Template

An example of the use of the Subsystem Template (Figure 16) is to decompose the event "No Pause in Yellow When Light Commanded From Green to Red" from the left branch of the example OR Template (Figure 7). At this point, the entire system is considered when determining which subsystems to decompose based upon this event. Subsequent decompositions using the Subsystem Template would use the scope of the subsystem that is currently being evaluated as the root subsystem of the template.

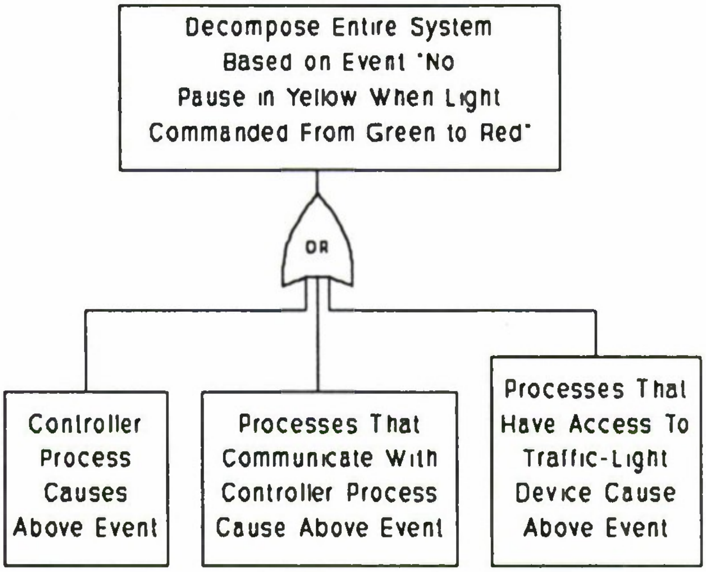

Decompose Entire System Based on Event "No Pause in Yellow When Light Commanded From Green to Red"
OR
Controller Process Causes Above Event
Processes That Communicate With Controller Process Cause Above Event
Processes That Have Access To Traffic-Light Device Cause Above Event

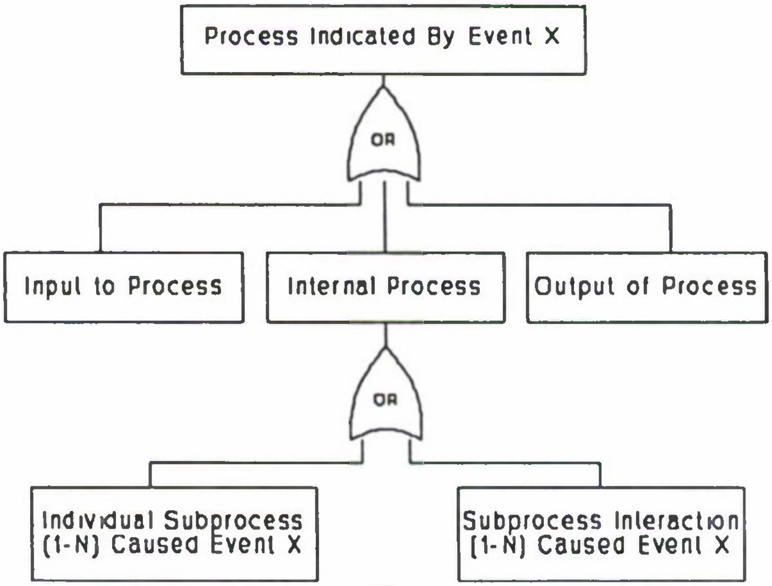
Figure 16. Example Subsystem Template Use

To evaluate the event "No Pause in Yellow When Light Commanded From Green to Red" for subsystem decomposition, the analyst must determine from the system design the processes that control the traffic light, the subsystem(s) that communicate with these controlling processes, and the processes that actually have access to the traffic light. The processes that can be decomposed from the system are determined by evaluating the High-Level Traffic-Light Design (Figure 14).

As shown in Figure 16, the "Controller" process is indicated in the left branch of the template because it is the process within the system that controls the traffic light by sending commands to set the traffic signal to the "Set Signal" process. The middle branch of Figure 16 indicates that all the processes that communicate with the "Controller" process must be considered by the decomposition. The right branch of Figure 16 indicates that all processes that actually have access to the traffic light should also be considered in this decomposition step.

2. Process Template

The Process Template (Figure 17) is used to decompose the subsystem by evaluating the three basic areas of the subsystem. These three areas are distinguishable by their relation to the overall process contained within the subsystem.


Process Indicated By Event X
OR
Input to Process
Internal Process
Output of Process
OR
Individual Subprocess (1-N) Caused Event X
Subprocess Interaction (1-N) Caused Event X

PROCESS ITSELF OR INTERACTION B/T PROCESS

Figure 17. Process Template

The first branch of the Process Template (Figure 17), Input To Process, decomposes the subsystem by considering the input to the subsystem from either a larger subsystem within which the current subsystem is contained, or input from the environment within which the subsystem operates. The input to the process within the subsystem is decomposed to evaluate whether or not input to the process can cause the hazard to occur.

The second branch of the Process Template (Figure 17), Internal Process, deals with the overall process that the current subsystem performs. As shown by the Process Template, the overall process itself can be decomposed into subprocesses 1-N. The total number of subprocesses (N) is limited because of the safety critical nature of the system being analyzed. If N is excessively large, then the system is inherently unsafe because the safety critical processes cannot reliably be isolated. The Process Template is designed for use on systems with a relatively small number of safety critical subprocesses. These N subprocesses are considered in two ways, individually and in combination.

Under the Individual Subprocess branch of the Process Template (Figure 17), each subprocess is considered individually to determine if any one of the N subprocesses can by itself result in the occurrence of the hazard. Each subprocess is strictly evaluated in isolation, with no interaction allowed from any other subprocess. If a subprocess always requires interaction with other subprocesses, then the individual subprocess decomposition is stopped. A subprocess requiring interaction is evaluated under the Subprocess Interaction branch of the Process Template.

The second way in which the Process Template (Figure 17) considers the N subprocesses that are decomposed via the Internal Process branch is through the evaluation of the interaction of one of the N subprocesses with another of the N subprocesses. The decomposition of the process of a subsystem that takes advantage of parallel processing must consider the different combinations of subsystems that are capable of processing in parallel. From this set of combinations, the software fault tree must consider whether any of the combinations are capable of interacting in a manner that would result in the occurrence of the hazard.

The third branch of the Process Template (Figure 17), Output of Process, decomposes the subsystem by considering the output that is produced by the overall process that the current subsystem performs. The output from the process is evaluated to consider whether the output can cause the hazard to occur.

An example of the use of the Process Template (Figure 18) is to decompose the "Controller" process of the high-level traffic light design. The decomposition starts with the analyst pursuing the "No Pause in Yellow When Light Commanded From Green to Red" event branch of the example OR Template (Figure 7). The analyst determines that the "Controller" process is a process that could be indicated by the occurrence of this event.

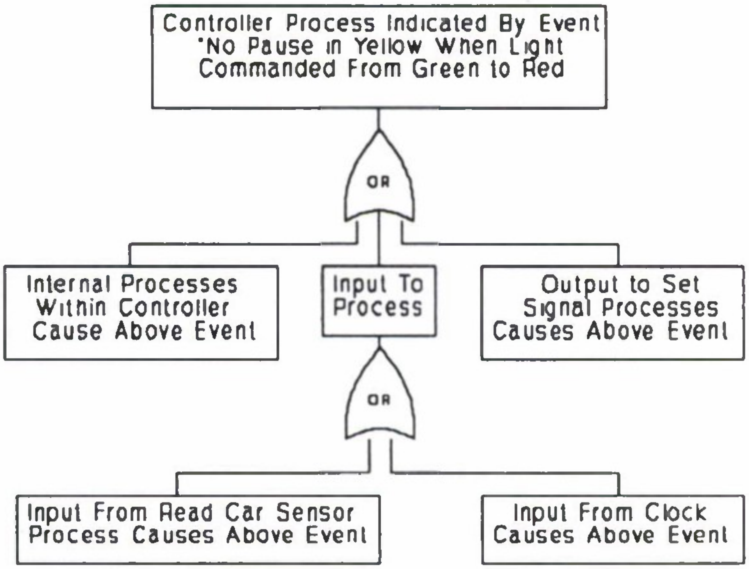

Controller Process Indicated By Event "No Pause in Yellow When Light Commanded From Green to Red
OR
Internal Processes Within Controller Cause Above Event
Input To Process
Output to Set Signal Processes Causes Above Event
OR
Input From Read Car Sensor Process Causes Above Event
Input From Clock Causes Above Event

Figure 18. Example Process Template Use

The "Controller" process is decomposed into three areas as shown in Figure 18. The first branch of the Process Template indicates that the internal processes within the "Controller" process should be analyzed. The second branch "Input to Process" indicates that the four "Read Car Sensor" processes and the "Clock" process should be analyzed since these processes provide input to the "Controller" process. The third branch indicates that the output from the "Controller" process to the four "Set Signal" processes should be analyzed to determine if the event in question could be a result of what is output by the "Controller" process.

3. Communication Template

The Communication Template (Figure 19) is used to decompose a subsystem based upon the communication between the processes that make up the subsystem. Processes can communicate via a rendezvous (as in Ada), through shared memory, or through the exchange of data.

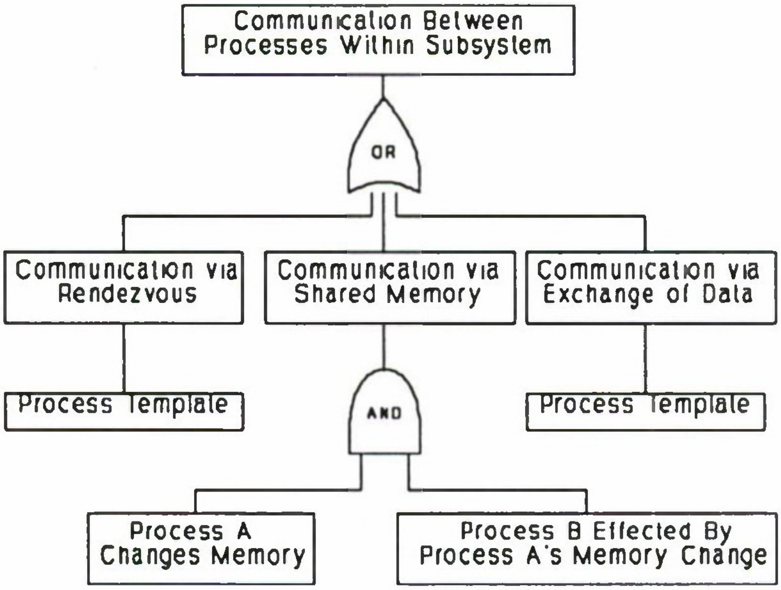

Communication Between Processes Within Subsystem
OR
Communication via Rendezvous
Communication via Shared Memory
Communication via Exchange of Data
Process Template
AND
Process Template
Process A Changes Memory
Process B Effected By Process A's Memory Change

Figure 19. Communication Template

The communication that can occur through a Ada rendezvous is checked using the Rendezvous branch of the Communication Template to determine if the hazard can occur here. Processes that communicate through rendezvous are analyzed in a manner similar to the way in which processes communicating through the exchange of data are analyzed. The two processes that rendezvous in effect join to form one overall process. This combined overall process is then analyzed using the Process Template.

For processes that communicate through shared memory, each process is analyzed under the Shared Memory Branch of the Communication Template with consideration given to changes to the shared memory that the first process can make. These changes may cause the second process to function in a manner such that the hazard occurs. As with the Process Template, the number of safety critical processes that communicate through shared memory must be small. If the number of safety critical processes communicating through shared memory is excessively large, then the system is inherently unsafe because the safety critical processes cannot reliably be isolated.

Whether or not the hazard can occur by way of the exchange of data through output from one process and input to another process is considered through the use of Process Templates. In the Exchange of Data branch of the Communication Template (Figure 19), the two processes exchanging data are considered joined together to form a single combined process. This combination process is then evaluated under the Process Template. The data that is output from the first process is analyzed using the Output of Process branch, while the data that is exchanged as input is evaluated using the Input to Process branch.

An example of the Communication Template (Figure 20) is limited by the simplicity of the example high-level traffic-light design. At this level, neither the

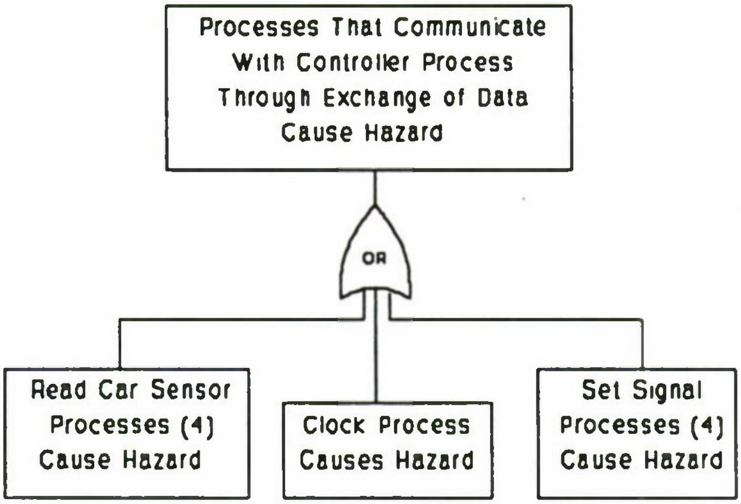

Processes That Communicate With Controller Process Through Exchange of Data Cause Hazard
OR
Read Car Sensor Processes (4) Cause Hazard
Clock Process Causes Hazard
Set Signal Processes (4) Cause Hazard

Figure 20. Example Communication Template Use

processes that communicate through rendezvous type operations, nor the processes that communicate through shared memory are known. The first and second branches of the Communication Template are therefore not pursued in the example.

The third branch of the Communication Template is used to ensure that all the processes that communicate with the "Controller" process are considered when analyzing the "Controller" process with regard to the hazard. As indicated by Figure 20, there are three distinct sets of processes that communicate with the "Controller" process through the exchange of data. These sets of processes are the four "Read Car Sensor" processes, the "Clock" process, and the four "Set Signal" processes. As shown by Figure 20, each of these distinct sets of processes must be analyzed for any role they might play in the "Controller" process causing the event, "No Pause in Yellow When Light Commanded From Green to Red".

4. Access Template

The Access Template (Figure 21) is used to decompose the subsystem by evaluating the access that the subsystem has to the control of a device or the maintenance of a database.

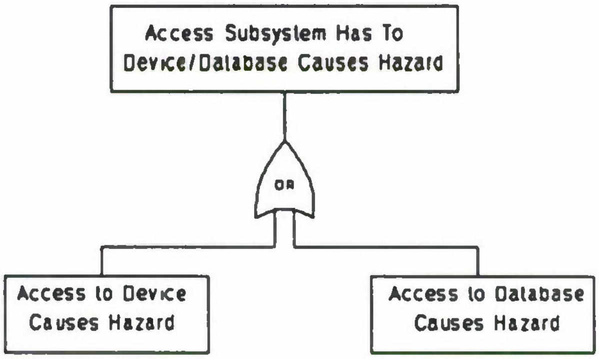

Access Subsystem Has To Device/Database Causes Hazard
OR
Access to Device Causes Hazard
Access to Database Causes Hazard

Figure 21. Access Template

Using the Access to Device branch of the Access Template, a subsystem controlling a device is evaluated to determine if the subsystem can control the device in a manner such that the hazard can occur. As shown in the Access To Device Template (Figure 22), the analysis of a subsystem's access to a device consists of three distinct evaluation conditions.

The first evaluation condition of the Access to Device Template is that the subsystem must have access to the device that causes the hazard. The second evaluation condition is that the subsystem must be capable of sending commands to the device, with the hazard occurring as a result of the device receiving these commands. The third evaluation condition is that the subsystem actually sends the hazard-causing commands to the device. These evaluation conditions are AND'd together because in order for the subsystem to cause the hazard through its access to the device, all three of these conditions must be met.

In a similar manner, the Access to Database branch of the Access Template (Figure 23) is used to determine if a subsystem maintaining a database can make entries or updates to the database that can result in a process accessing the database values and utilizing these values in such a manner that the hazard occurs. As shown in the Access To Database Template, the analysis of a subsystem's access to a database is evaluated using three distinct conditions.

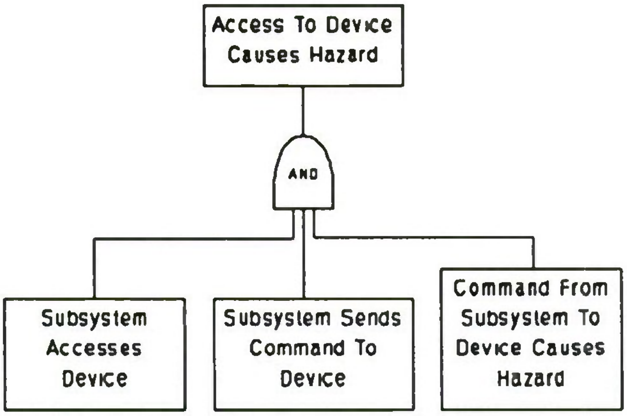

Access To Device Causes Hazard
AND
Subsystem Accesses Device
Subsystem Sends Command To Device
Command From Subsystem To Device Causes Hazard

Figure 22. Access To Device Template

The first evaluation condition is that the subsystem must have write access to the database from which values that cause the hazard are referenced.

The second evaluation condition is that the subsystem must be capable of writing values to the database, with the hazard occurring as a result of a process referencing these values. Here, the removal of a value from the database and having

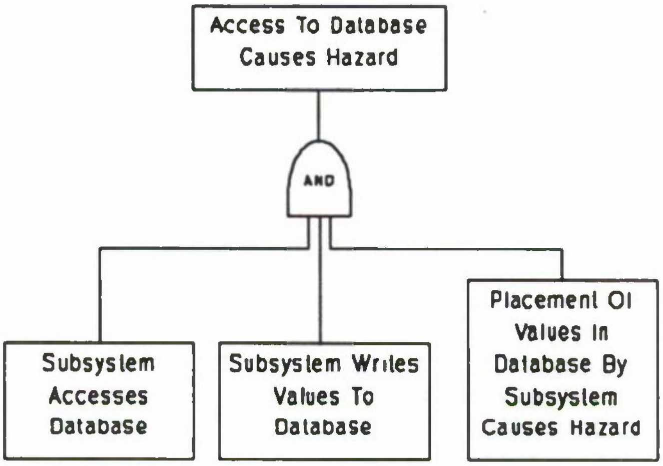

Access To Database Causes Hazard
AND
Subsystem Accesses Database
Subsystem Writes Values To Database
Placement Of Values In Database By Subsystem Causes Hazard

Figure 23. Access To Database Template

the hazard occur as a result of this removal is considered as an overwrite of an original valid value.

The third evaluation condition is that the subsystem actually writes the hazard-causing value to the database. These evaluation conditions are AND'd together because in order for the subsystem to cause the hazard through its access to the database, all three of these conditions must be met.

An example of the use of the Access To Device branch of the Access Template is the decomposition of the Output of Process branch of the Process Template example (Figure 17). This branch indicates that the "Set Signal" processes need to be analyzed to determine if the "No Pause in Yellow When Light Commanded From Green to Red" event could be caused within these processes. The Example Access To Device Template (Figure 24) shows how a subsystem causing the "No Pause in Yellow" hazard is considered.

Each "Set Signal" process has access to a traffic light since these processes control the changing of the traffic lights, so these processes are indicated in the first

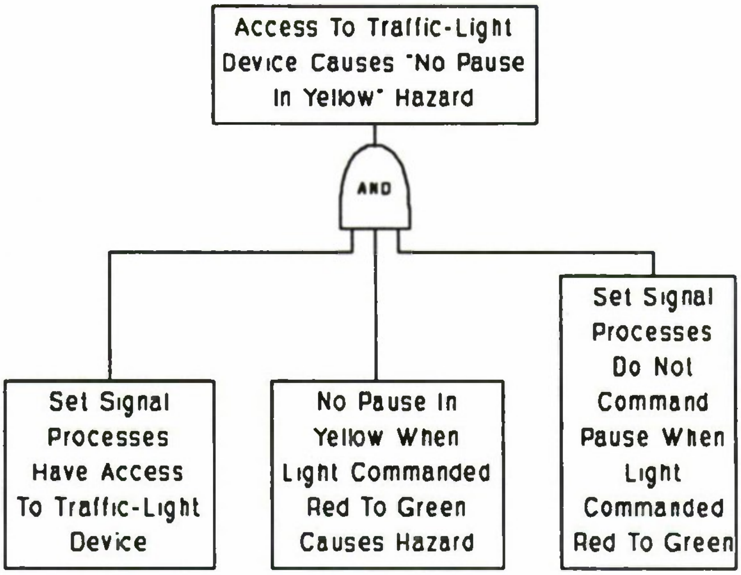

Access To Traffic-Light Device Causes "No Pause In Yellow" Hazard
AND
Set Signal Processes Have Access To Traffic-Light Device
No Pause In Yellow When Light Commanded Red To Green Causes Hazard
Set Signal Processes Do Not Command Pause When Light Commanded Red To Green

Figure 24. Example Access To Device Template Use

branch of the example. In the second branch of the example, the commands that these processes give to the traffic light must be analyzed to determine if it is possible for these processes to send commands that could cause, in this example, the light to not pause in yellow when changing from green to red. The third branch of the example indicates that the hazard-causing commands indicated by the second branch must actually be sent to the traffic light in order for the hazard to occur.

H. TIME RELATIONSHIP OF EVENTS

The decomposition of the hazard event carries with it the implication of going back in time. As an event is decomposed, all the breakdowns that are the result of the decomposition have the requirement of occurring simultaneously with the hazard, or at some time prior to the occurrence of the hazard. SFTA must go backward in time from the occurrence of the hazard so that the system can be established in a normal safe operational state. The implication of the fault tree being backed out all the way to the initialization of the system without ending in a safe operational state is that the unsafe event is not prevented by the system.

The Time Template (Figure 25) shows an event that is decomposed into an event that must occur first in order for the later event to occur. This decomposition is based upon the logical implication, "if A implies B and B implies C, then A implies C", and carries with it the relationship of the events with regard to time. In Figure 25, Event 1 is the event that the fault tree indicates would cause the occurrence of the hazard. Event 2 is an event that causes Event 1 to occur. The link between the two events is that Event 2 must occur before Event 1. Although Event 2 by itself does not directly cause the hazard to occur, Event 2 must be considered due to its relationship with Event 1. Because Event 2 causes Event 1 to occur, Event 2 indirectly causes the occurrence of the hazard.

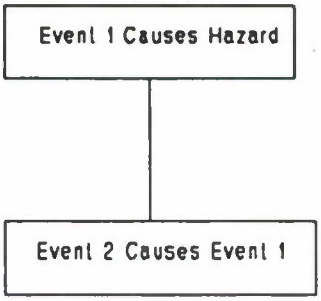


Figure 25. Time Template

An example of the use of the Time Template (Figure 26) can be found by modifying the requirements of the traffic light control system. The modification is that one of the roads leading to the intersection has a blind curve that must be driven through just prior to entering the intersection. On the curve prior to the intersection is a flashing yellow light that serves to warn motorists that an intersection with a traffic light lies ahead.

Figure 26 shows how the Time Template can be used to decompose an event into a second event that precedes and causes the first event to occur. The event that causes the "Perpendicular Cars" hazard is a car entering the intersection against a traffic light that is displaying a red signal. The event of entering the intersection against a red light could indirectly be caused by the flashing yellow warning light being inoperative. Without the warning to slow down on the blind curve prior to the intersection, the motorist does not have enough room to stop the vehicle, given reaction time and stopping distances for a car traveling at normal speed. Thus, the failure of the flashing yellow warning light indirectly causes the "Perpendicular Cars" hazard.

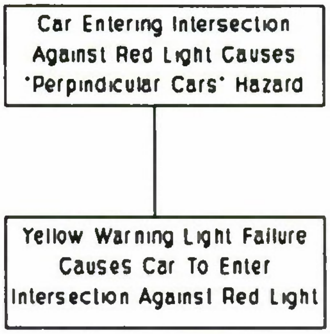

Car Entering Intersection Against Red Light Causes 'Perpindicular Cars' Hazard
Yellow Warning Light Failure Causes Car To Enter Intersection Against Red Light

Figure 26. Example Time Template Use

I. INTERDEPENDENCE OF DECOMPOSITION DIMENSIONS

The two decomposition dimensions, Specificity-of-Event and Subsystem-Size, are dependant on each other. Both are goals of the hazard decomposition. The specificity of an event can be made more specific as a means of aiding in the decomposition of a subsystem's size. In a similar manner, the decomposition of a subsystem's size serves as a means of making the targeted event more specific. As the Decomposition Dimensions in Figure 5 at the beginning of the chapter shows, the decomposition of a hazard occurs along two distinct dimensions, specificity of event and subsystem size. As the system is analyzed, either decomposition dimension can be used as a means of decomposing the hazard. The templates presented in this chapter form a framework from which it is possible to decompose the hazard along either of the two dimensions. As the hazard analysis proceeds, it may be advantageous to decompose the hazard or event with the intent of making the other decomposition dimension more useful. In other words, to advance the Subsystem-Size decomposition, the Specificity-of-Event decomposition dimension may be invoked specifically to help the analyst advance the Subsystem-Size decomposition. In the following chapter, the framework of decomposition dimensions and decomposition templates presented in this chapter are applied to a more complicated example.

III. APPLICATION OF DECOMPOSITION TEMPLATES

A. ONE-LANE BRIDGE REQUIREMENTS MODEL

In this chapter, the decomposition templates presented in Chapter II are applied to the requirements model of a software system developed by Ripps [Ref. 10:p. 32] in his work on the development of real time requirements. This software system is being designed to control the traffic flow on a one-lane bridge. As depicted by Figure 27, the one-lane bridge is shared by both lanes of a dual-lane road. The software system controls access to the bridge through the use of traffic lights, and is to keep traffic travelling across the bridge flowing smoothly in one direction or the other as appropriate.

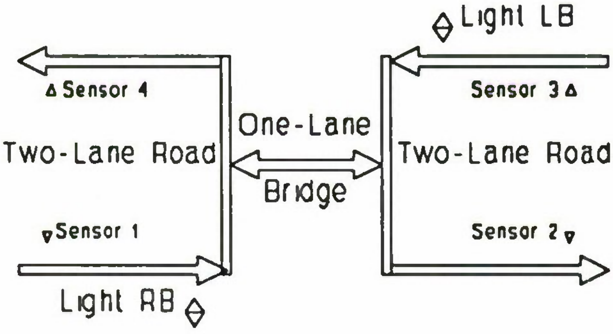


Figure 27. Depiction of One-Lane Bridge

The requirements model for the one-lane bridge appears in Appendix A as a set of rules that governs the software system controlling the bridge. Since this behavioral model is somewhat difficult to comprehend as a set of rules, a graphical representation of the one-lane bridge control system is provided in Appendix B.

B. ONE-LANE BRIDGE HAZARD

For the purpose of presenting an application of the decomposition templates, the one-lane bridge control system is analyzed for the hazard "Two Cars Simultaneously on Bridge in Opposite Directions". Normally, the event at the root of a fault tree would be much more specific than this event, but for the purposes of illustration an event of this generality is used so as to demonstrate the various applications of the decomposition templates developed in Chapter II. The event "Two Cars Simultaneously on Bridge in Opposite Directions" is used as a starting point because it is an event that is unsafe for the motorists travelling across the bridge, and is an event that the software control system should not allow to occur. This hazard appears at the root of the resultant software fault tree, and is the initial event decomposed via the decomposition templates. It is noted that the decomposition process is not concerned with whether the hazard could occur for reasons beyond the control of the software, such as cars ignoring a red light and illegally entering the bridge.

C. DECOMPOSITION OF ONE-LANE BRIDGE HAZARD

1. Specificity-of-Event Decomposition Dimension

The Specificity of Event decomposition dimension is used to make the hazard event, "Two Cars Simultaneously on Bridge in Opposite Directions", more specific in terms of events and conditions associated with these events. Inspection of the hazard indicates that one way in which the hazard event could be decomposed is by using the AND (WHILE/WHEN) template (Figure 28).

The AND (WHILE/WHEN) template is indicated because the hazard requires that an event occurs while a condition holds. The event is "Lights in Both Directions Green Simultaneously". The condition associated with this event is that "Two Cars Approach From Opposite Directions". All subsequent decompositions under the event side of the AND (WHILE/WHEN) template branch occur within the scope of this condition.

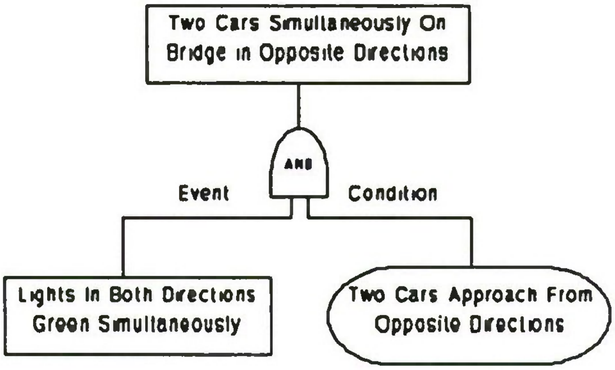

Two Cars Simultaneously On Bridge in Opposite Directions
AND
Event
Condition
Lights In Both Directions Green Simultaneously
Two Cars Approach From Opposite Directions

Figure 28. Bridge Hazard Decomposition

The decomposition of the event side of Figure 28 makes further use of the Specificity of Event decomposition dimension and is based upon the analyst's knowledge of the system. Inspection of the requirements model in Appendix B reveals that there are four separate system controlled events that could cause the event side of Figure 28, "Lights in Both Directions Green Simultaneously", to occur. Since these four events are independent of one another, they are decomposed from the event side of Figure 28 through the use of the OR template (Figure 29). Each of these four events are analyzed independently by the fault tree because any one alone could cause the event side of Figure 28 to contribute to the occurrence of the hazard.

The events decomposed by Figure 29 come from the requirements model for the one-lane bridge (Appendix B). The analyst must determine the ways in which the event "Lights in Both Directions Green Simultaneously" from Figure 28 could occur within the scope of the system's control. The requirements model's "Initialization"

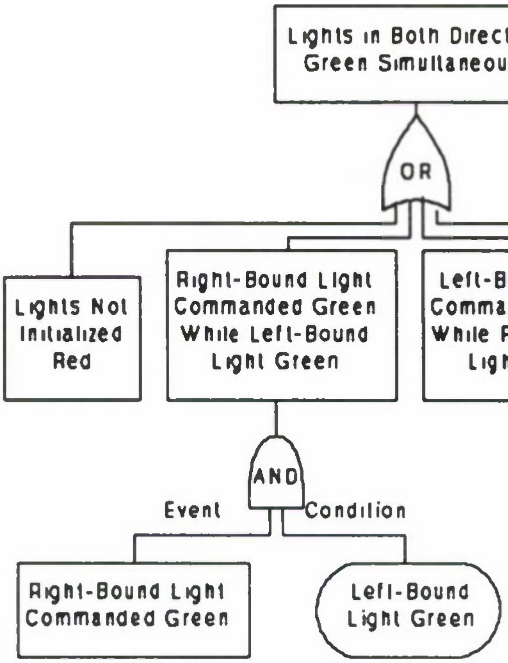

Lights in Both Directions Green Simultaneously
OR
Lights Not Initialized Red
Right-Bound Light Commanded Green While Left-Bound Light Green
Left-Bound Light Commanded Green While Right-Bound Light Green
Software Commands Both Lights Green Simultaneously
AND
Event
Condition
Right-Bound Light Commanded Green
Left-Bound Light Green

Figure 29. Decomposition of Bridge Hazard Event

process and two light control processes indicate that the "Lights in Both Directions Green Simultaneously" event could occur by at least four events that are controllable by the system's software.

The first event is "Lights Not Initialized Red". If the lights are not initialized red, then the possibility exists for the lights to initially come on green when the system starts to control the bridge.

The second and third events consider the cases where one light is commanded to green while the other light already green. These two events are essentially the same event with the difference being which light is already green, and which light is being commanded to green by the software. Due to this reflexive condition, only the event "Right-Bound Light Commanded Green While Left-Bound Light Green" will be expanded. The conditional nature of this event indicates that the event "Right-Bound Light Commanded Green While Left-Bound Light Green" can be further decomposed using the AND (WHILE/WHEN) template.

The fourth event considered that is controllable by the system software is "Software Commands Both Lights Green Simultaneously". This event delves into whether or not the system is capable of turning the traffic lights in both directions green at the same time.

As shown in Figure 29, the event "Right-Bound Light Commanded Green While Left-Bound Light Green" is decomposed into the event "Right-Bound Light Commanded Green" and the condition "Left-Bound Light Green" through the application use of another AND (WHILE/WHEN) template. This decomposition step is necessary to distinguish the event being considered for further decomposition by the fault tree from the condition whose scope will cover all subsequent decompositions of the associated event.

Cases such as one or both lights being stuck in the green position are hardware related and are not considered by this particular fault tree application. The analyst may elect to include hardware events as a means of branching into hardware fault tree analysis in a manner similar to that presented by Leveson [Ref. 5:p. 570]. If this were the case, then the inclusion of hardware controlled events such as lights being stuck green would be appropriate in this decomposition step.

2. Subsystem-Size Decomposition Dimension

Up until this point, the decomposition process has used the Specificity-of-Event decomposition dimension for decomposing the hazard. Further decomposition of the event branches of Figure 29 with regard to the Specificity-of-Event dimension do not readily follow, suggesting that the Subsystem-Size decomposition dimension should be considered. To illustrate the Subsystem-Size decomposition dimension, the event branches of Figure 29 are further decomposed using Subsystem decomposition templates.

The Subsystem template allows the current subsystem to be broken down into subsystems of smaller scope that are more directly applicable to the occurrence of the hazard. These decomposition steps are driven by the events that are being decomposed and start with the entire system being considered as the root of the Subsystem template. Each event from the branches of Figure 29 decomposes in this case the entire system into the subsystem that plays a role in each particular events occurrence.

The event "Lights Not Initialized Red" from Figure 29 indicates that the further decomposition of this event requires the system to be broken down into the subsystem that controls the initialization of the lights, in this case the "Initialize System" process (Figure 30). This decomposition step makes use of the process branch of the Subsystem template, indicating that it is the input-process-output of the "Initialize System" process that should be next analyzed to determine whether the lights are initialized red by the system. Neither the Communication branch nor the Access branch of the Subsystem Template are indicated as avenues for further decomposition because the requirements model in Appendix A presents "Initialize System" as a process that does not communicate with other processes and does not have access to any device or database. Therefore, the Communication and Access branches are both represented by a diamond indicating that there is no further decomposition along these avenues.

The application of the Process template to the process branch of Figure 30 is represented by Figure 31. The process "Initialize System" is decomposed with regard to the event "Lights Not Initialized Red" carried through from the subsystem decomposition in Figure 30. The Process template gives a structure for analyzing the

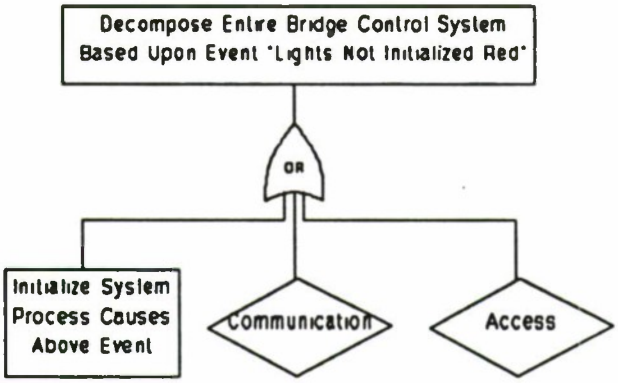

Decompose Entire Bridge Control System Based Upon Event "Lights Not Initialized Red"
OR
Initialize System Process Causes Above Event
Communication
Access

Figure 30. Process Subsystem Decomposition

process contained within "Initialize System". Figure 31 shows that the input to the process, the output from the process, and the constructs internal to the process must all be analyzed to determine if one or more of these parts of the "Initialize System" process could allow the "Lights Not Initialized Red" event to occur.

The event "Right-Bound Light Commanded Green" from Figure 29 indicates that the access that the system has to the right-bound light should be checked for this event. By analyzing the graphical representation of the one-lane bridge requirements model in Appendix B, it is apparent that the part of the system that has access to the right-bound light is the "Control RB Light" process. Figure 32 indicates that the access of the "Control RB Light" process to the right-bound light should next be analyzed to determine whether the right-bound light can be commanded green while the condition of the left-bound light already being green holds. Neither the Process branch nor the Communication branch of the Subsystem template are indicated for further decomposition, so both of these branches are represented as diamonds in Figure 32.

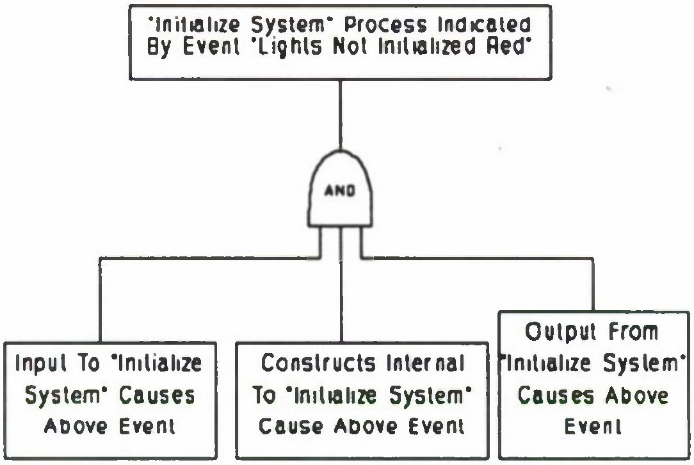

'Initialize System' Process Indicated By Event 'Lights Not Initialized Red'
AND
Input To 'Initialize System' Causes Above Event
Constructs Internal To 'Initialize System' Cause Above Event
Output From 'Initialize System' Causes Above Event

Figure 31. Process Template

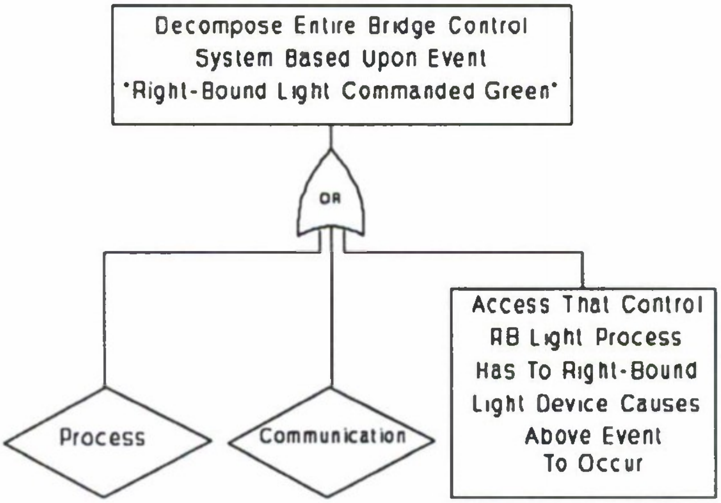

Decompose Entire Bridge Control System Based Upon Event 'Right-Bound Light Commanded Green'
OR
Process
Communication
Access That Control RB Light Process Has To Right-Bound Light Device Causes Above Event To Occur

Figure 32. Access Subsystem Decomposition

The application of the Access template to the access branch Figure 32 is represented by Figure 33. The process "Control RB Light" is decomposed with regard to the event "Right-Bound Light Commanded Green" carried through from the subsystem decomposition in Figure 32. The Access template gives a structure for analyzing the access that the process "Control RB Light" has to the right-bound light of the one-lane bridge.

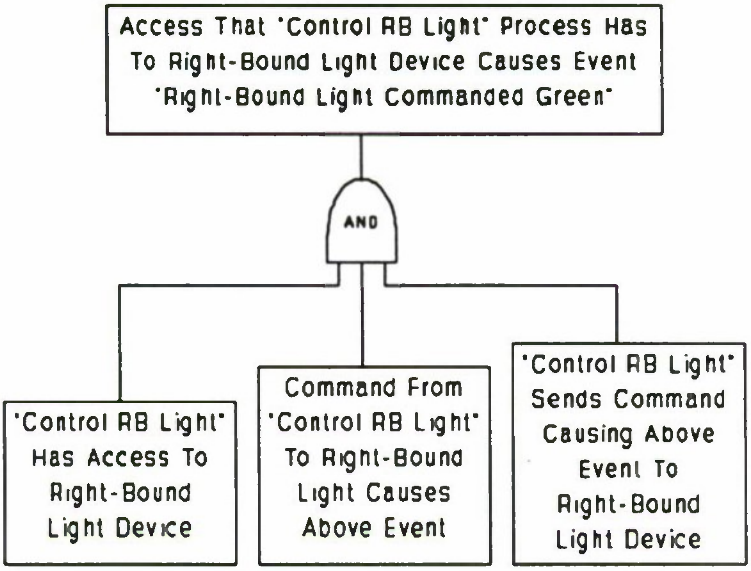
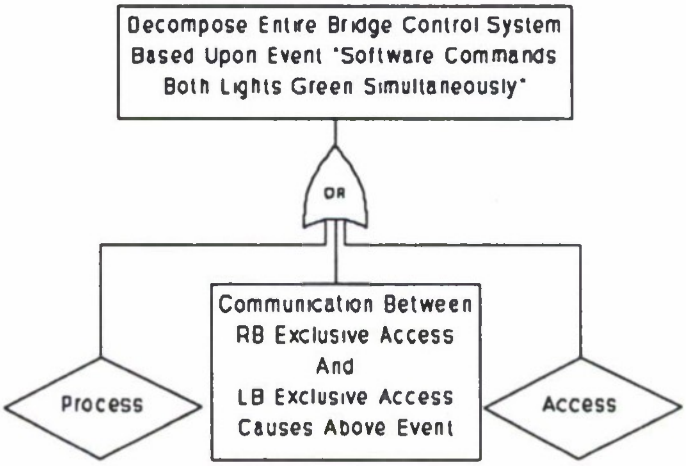

Access That 'Control RB Light' Process Has To Right-Bound Light Device Causes Event 'Right-Bound Light Commanded Green'
AND
'Control RB Light' Has Access To Right-Bound Light Device
Command From 'Control RB Light' To Right-Bound Light Causes Above Event
'Control RB Light' Sends Command Causing Above Event To Right-Bound Light Device

Figure 33. Access Template

The template in Figure 33 shows the three cases that must hold, and therefore be analyzed by the fault tree, in order for the "Control RB Light" process to cause the event "Right-Bound Light Commanded Green". The first case is that "Control RB Light" must have access to the Right-Bound light on the bridge. The second case is that "Control RB Light" must be capable of sending one or more commands to the Right-Bound light that would result in the event "Right-Bound Light Commanded Green". The third case is that "Control RB Light" must actually send the commands specified in the second case to the Right-Bound light.

The decomposition of the event "Software Commands Both Lights Green Simultaneously" from Figure 29 requires knowledge of how the system allows each light to be commanded green. Requirements model rules 1 and 5 from the requirements model in Appendix A require each light control process to wait until exclusive access to the bridge is granted before turning the controlled light green. This requirement of waiting for exclusive access indicates that the system should be broken down into the subsystems that communicate through whatever process controls the exclusive access to the bridge (Figure 34).



Decompose Entire Bridge Control System Based Upon Event "Software Commands Both Lights Green Simultaneously"
OR
Process
Communication Between RB Exclusive Access And LB Exclusive Access Causes Above Event
Access

Figure 34. Communication Subsystem Decomposition

The requirements model identifies two processes that control the access to the bridge through a rendezvous type operation. These two processes are the "RB Wait For Exclusive Access To Bridge" process and the "LB Wait For Exclusive Access To Bridge" process. For illustration purposes these two processes will be referred to as "RB Exclusive Access" and "LB Exclusive Access" respectively. The application of the Communication template to the Communication branch of Figure 34 is represented by Figure 35. The two processes, "RB Exclusive Access" and "LB Exclusive Access", that control the access to the bridge through a rendezvous type operation are decomposed with regard to the event "Software Commands Both Lights Green Simultaneously" carried through from the subsystem decomposition in Figure 34. The Communication template gives a structure for analyzing the rendezvous type operation which is the way the two exclusive access processes are assumed to communicate. At this point, neither the Process branch nor the Access branch of the Subsystem template are indicated for further decomposition. These branches are therefore both represented by diamonds in Figure 34.

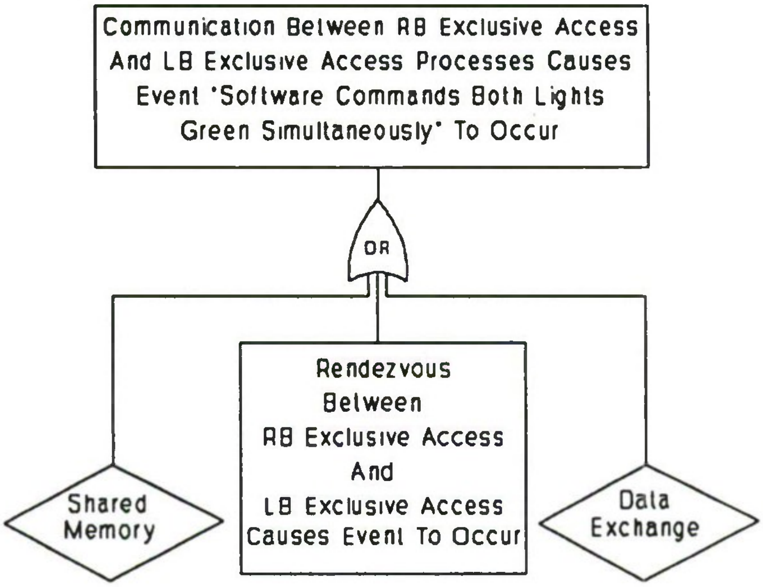

Communication Between RB Exclusive Access And LB Exclusive Access Processes Causes Event 'Software Commands Both Lights Green Simultaneously' To Occur
OR
Shared Memory
Rendezvous Between RB Exclusive Access And LB Exclusive Access Causes Event To Occur
Data Exchange

Figure 35. Communication Template

As Figure 35 indicates, the rendezvous between the two exclusive access processes must be evaluated. The effect of a rendezvous is to create a new process. This new process is entirely made up of the interaction that goes on between the two exclusive access processes when the rendezvous type operation is accomplished, and is in fact the single process that actually controls exclusive access to the bridge. This new process must be analyzed to determine if it could cause the event "Software Commands Both Lights Green Simultaneously". The next step of the analysis, as indicated by the Communication template, is to analyze the new process using a Process Template. Based upon the assumption that the two exclusive access processes communicate via a rendezvous type operation, the Communication via Shared Memory and Communication via Exchange of Data branches of the Communication template are not pursued. These two branches are therefore each represented by a diamond in the Communication Template of Figure 35.

3. Overall Fault Tree for Bridge Hazard

Figure 36 shows the entire software fault tree that results from the application of decomposition templates to the one-lane bridge hazard. The decomposition process using the decomposition templates developed in Chapter II continues until the point at which the specific system modules, events and associated conditions are indicated is sufficient detail. At this point, if the software system has been developed down to the code level, the software fault tree analysis can be continued through the application of Cha statement templates. If the software system is has not yet been developed to the code level, the modules, events and conditions indicated by the software fault tree can be used to provide specific input regarding what the detailed design of the indicated modules should be concerned with.

D. SUMMARY OF DECOMPOSITION TEMPLATE APPLICATION

Hazard decomposition using the decomposition templates presented in this thesis continues until a contradiction is reached, or a specific process within the software system is indicated. Where a specific process is indicated, the decomposition of the hazard continues with the use of statement templates. The use of statement templates is supported by the structure of the fault tree resulting from the application of the

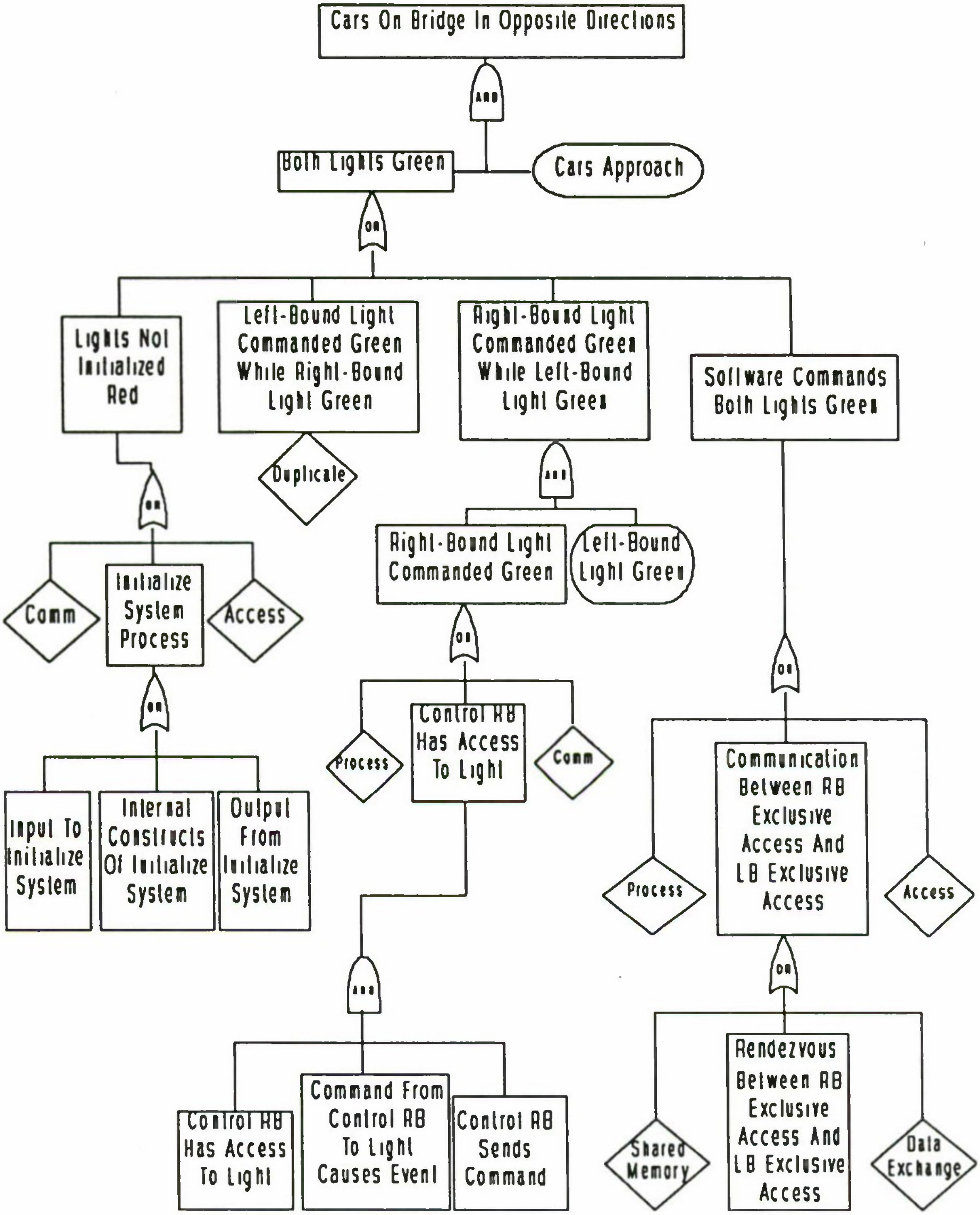

Cars On Bridge In Opposite Directions
AND
Both Lights Green
Cars Approach
OR
Lights Not Initialized Red
Left-Bound Light Commanded Green While Right-Bound Light Green
Right-Bound Light Commanded Green While Left-Bound Light Green
Software Commands Both Lights Green
Duplicate
AND
OR
Right-Bound Light Commanded Green
Left-Bound Light Green
Comm
Initialize System Process
Access
OR
Process
Control RB Has Access To Light
Comm
OR
OR
Process
Communication Between RB Exclusive Access And LB Exclusive Access
Access
Input To Initialize System
Internal Constructs Of Initialize System
Output From Initialize System
AND
OR
Control RB Has Access To Light
Command From Control RB To Light Causes Event
Control RB Sends Command
Shared Memory
Rendezvous Between RB Exclusive Access And LB Exclusive Access
Data Exchange

Figure 36. Overall Fault Tree for Bridge Hazard

decomposition templates. The fault tree up to this point contains the conditions that must hold and the events that must occur in order for the system to arrive in the unsafe state of the hazard. With the fault tree indicating a specific process or subsystem, the use of statement templates can be better focused. The statement templates now have the advantage of starting off pointing to the specific process to be analyzed, with both the events and associated conditions known.

IV. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The purpose of this thesis is to develop a formalized method of decomposing system-level hazards. The decomposition templates presented herein provide the software fault tree analyst with a framework for decomposing a hazard to the point at which line by line code analysis can be conducted with existing statement templates. This framework serves as a formal method for conducting the decomposition of system-level hazards, and ensures that as many as possible of the applicable decomposition aspects are considered. The decomposition of system-level hazards had previously been conducted in a largely human intensive manner, carrying both the drawback of introducing human error in the form of oversight as well as the strength of human insight. The application of the decomposition templates developed in this thesis serves to reduce the former and enhance the later.

1. Relevance to MIL-STD-882B

Task 202 of MIL-STD-882B [Ref. 11:p. 202-1] provides guidelines for the identification of safety critical areas and the evaluation of hazards. The hazards identified here include the potential contribution of software events to system/subsystem mishaps. These software events include software commands and responses such as inadvertent commands, failure to command, and untimely commands and responses. The decomposition templates developed within the Specificity-of-Event Decomposition Dimension of Chapter II can be used to determine the software events that can contribute to system mishaps. The events identified by the decomposition templates can then be incorporated into the safety design criteria of the software specifications.

Task 203 of MIL-STD-882B [Ref. 11:p. 203-1] provides guidelines for the identification of hazards associated with the design of subsystems. The hazards identified here include the potential contribution of software events and faults with regard to the safety of a subsystem. The subsystem decomposition templates developed within the Subsystem-Size Decomposition Dimension of Chapter II can serve as an analysis technique for performing Task 203's Subsystem Hazard Analysis with regard to the software of a system, and can be used to determine whether the safety design criteria in the software specification have been satisfied.

2. Advantages of Decomposition Templates

The advantages of using the decomposition templates presented in this thesis to decompose system-level hazards stem from the formalized standpoint from which the templates were developed. The decomposition templates give the software fault tree analyst a structured viewpoint from which to evaluate the software system. The fault tree analyst can shift between the two interdependent decomposition dimensions as necessary, focusing on either increasing the specificity of the event being considered, or reducing the scope of the subsystem associated with the event. Shifting from one decomposition dimension to another essentially allows the decomposition process to shift the perspective from which the software system is being considered. The templates serve to aid the analyst by providing a step by step framework with which to approach each subsequent decomposition of the system-level hazard. As indicated in Chapter III, this decomposition process can be continued until the point at which statement templates can be used. The structure of the software fault tree provides the analyst with information that focuses the application of statement templates. The application of statement templates yields information of sufficient detail so as to allow the analyst to make use of the Ada exception handling mechanism as a method of pruning the developing fault tree. Pruning a fault tree in this manner is analogous to

Leveson and Harvey's method of inserting run-time checks into the code to trap a developing unsafe state [Ref. 4:p. 576]. Each decomposition step provides specific events and/or conditions that are to be further evaluated, as well as the specific subsystem within which to evaluate these events and conditions. Applying the decomposition templates presented in this thesis effectively yields pointers to specific modules within the overall software system, along with information such as events and conditions that the module needs to be evaluated for. The effect of the use of hazard decomposition templates is to give the application of statement templates a specific starting point, as well as a specific hazard to be analyzed by the statement templates.

3. Limitations of Decomposition Templates

The application of decomposition templates does not relieve the analyst from the need for a through understanding of the software system. In order for the Subsystem-Size Decomposition Dimension to be effective, the analyst must be thoroughly familiar with the communication links and interfaces through which the software system will function. The reliance of this decomposition dimension on the communication between modules implies that, at a minimum, a high-level design of the software system to be analyzed is needed in order for this decomposition dimension to be effective. This requirement obviously limits the effectiveness of this dimension if application when applied to solely the requirements specification of a system.

The effectiveness of the Specificity-of-Event Decomposition Dimension is also reliant on the analyst's knowledge of the software system. The focus of this decomposition dimension centers around itemizing each event that could cause the given hazard in the current subsystem. For the fault tree to fully reflect all the possible paths in which the hazard could occur, the scope of the decomposition must constitute a complete enumeration of the ways in which the hazard could occur within the current subsystem. In other words, the events resulting from any decomposition step must represent in total the ways in which the hazard could occur at that level. A problem arises in how confident the analyst is that every resultant event has been properly reflected in the decomposition. Specifically, the analyst must know when every event that could result from the decomposition of the hazard within the current subsystem has been considered. For the Specificity-of-Event Decomposition Dimension to be effective, it is clear that a through knowledge of how the software system functions is required.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

Several topics related to this thesis warrant further research. First, research is needed to determine how the process of decomposing a system-level hazard can be automated. Even with the software fault tree limited to the analysis of safety critical events, the fault tree of a moderately sized system will be substantial. Although aided by the decomposition template framework, the development of any fault tree is labor intensive. The reliance of the Subsystem-Size Decomposition Dimension on the communication links and interfaces of a system suggests that, given a high-level design, automation of this decomposition dimension may be possible.

Second, the issue of integrating formal requirements into a safety analysis should be investigated. The ability of the Specificity-of-Event Decomposition Dimension to be applied to the requirements of a system indicates that this decomposition dimension can be an effective method of analyzing the requirements of a system from a safety standpoint.

A third area recommended for research is the possibility of transferring the logical basis of the templates into a formal logic. A formal logic would enable the analyst to give proof conditions that could be used to formally prove whether the system could arrive in a state in which a specific hazard has occurred.

APPENDIX A. REQUIREMENTS MODEL RULES FOR ONE-LANE BRIDGE

Rule 1 : If in right-bound state Traffic_Idle and a right-bound car approaches the bridge (as detected by sensor 1), then set RB_Cars to 1, request Access_to_Bridge, and enter right-bound state Wait_for_Exclusive_Access to_Bridge. When exclusive access is granted, enter right-bound state Traffic_Active and turn light RB green.

Rule 2 : If in right-bound state Traffic_Active and a right-bound car leaves the bridge (as detected by sensor 2) and RB_Cars is not 1, then decrement RB_Cars by 1.

Rule 3 : If in right-bound state Traffic_Active and a right-bound car leaves the bridge (as detected by sensor 2) and RB_Cars is not 1, then decrement RB_Cars by 1.

Rule 4 : If in right-bound state Traffic_Active and a right-bound car leaves the bridge and RB_Cars is 1, then turn light RB red, set RB_Cars to 0, release Access_to_Bridge, and enter right-bound state Traffic_Idle.

Rule 5 : If in left-bound state Traffic_Idle and a left-bound car approaches the bridge (as detected by sensor 3), then set LB_Cars to 1, request Access_to_Bridge, and enter left-bound state Wait_for_Exclusive_Access_to_Bridge. When exclusive access is granted, enter left-bound state Traffic_Active and turn light LB green.

Rule 6 : If in right-bound state Traffic_Active and a right-bound car leaves the bridge (as detected by sensor 2) and RB_Cars is not 1, then decrement RB_Cars by 1.

Rule 7 : If in left-bound state Traffic_Active and a left-bound car leaves the bridge (as detected by sensor 4) and LB_Cars is not 1, then decrement LB_Cars by 1.

Rule 8 : If in left-bound state Traffic_Active and a left-bound car leaves the bridge and LB_Cars is 1, then turn light LB red, set LB_Cars to 0, release Access_to_Bridge, and enter left-bound state Traffic_Idle.

APPENDIX B. GRAPHICAL REPRESENTATION OF ONE-LANE BRIDGE

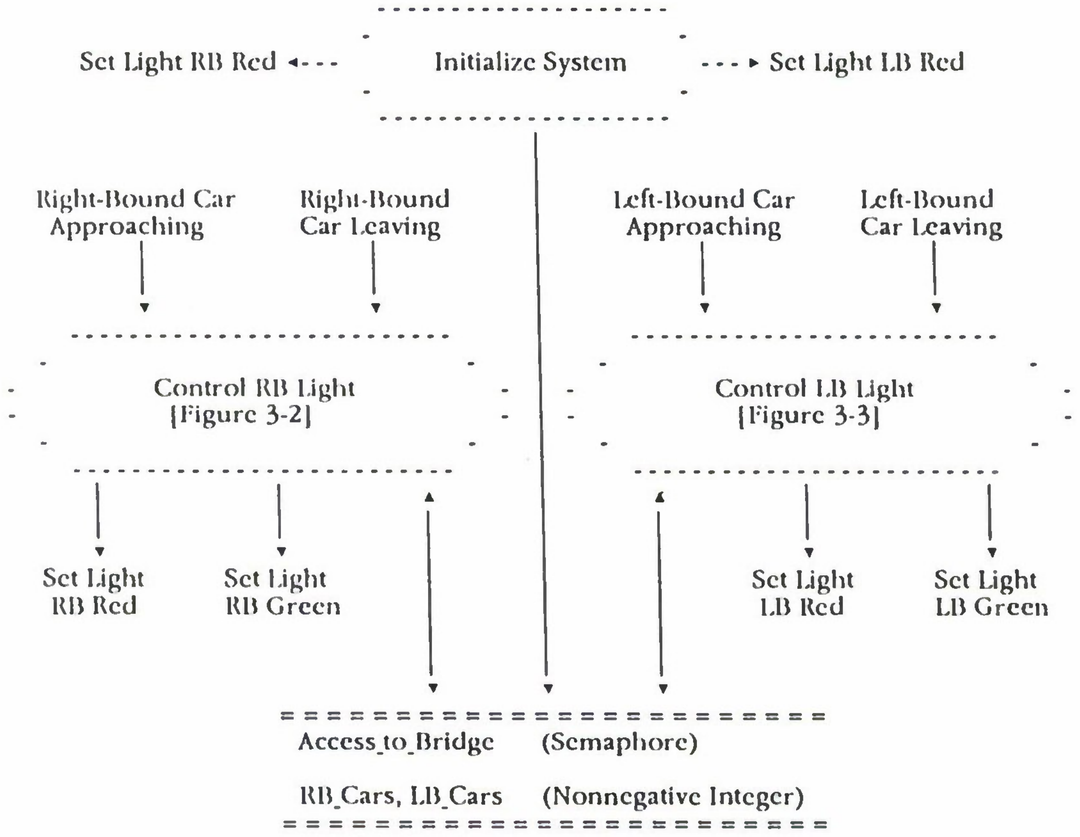

Set Light RB Red
Initialize System
Set Light LB Red
Right-Bound Car Approaching
Right-Bound Car Leaving
Left-Bound Car Approaching
Left-Bound Car Leaving
Control RB Light [Figure 3-2]
Control LB Light [Figure 3-3]
Set Light RB Red
Set Light RB Green
Set Light LB Red
Set Light LB Green
Access_to_Bridge (Semaphore)
RB_Cars, LB_Cars (Nonnegative Integer)

FIGURE 3-1: Graphical Representation of Behavioral Requirements: Control Transformations

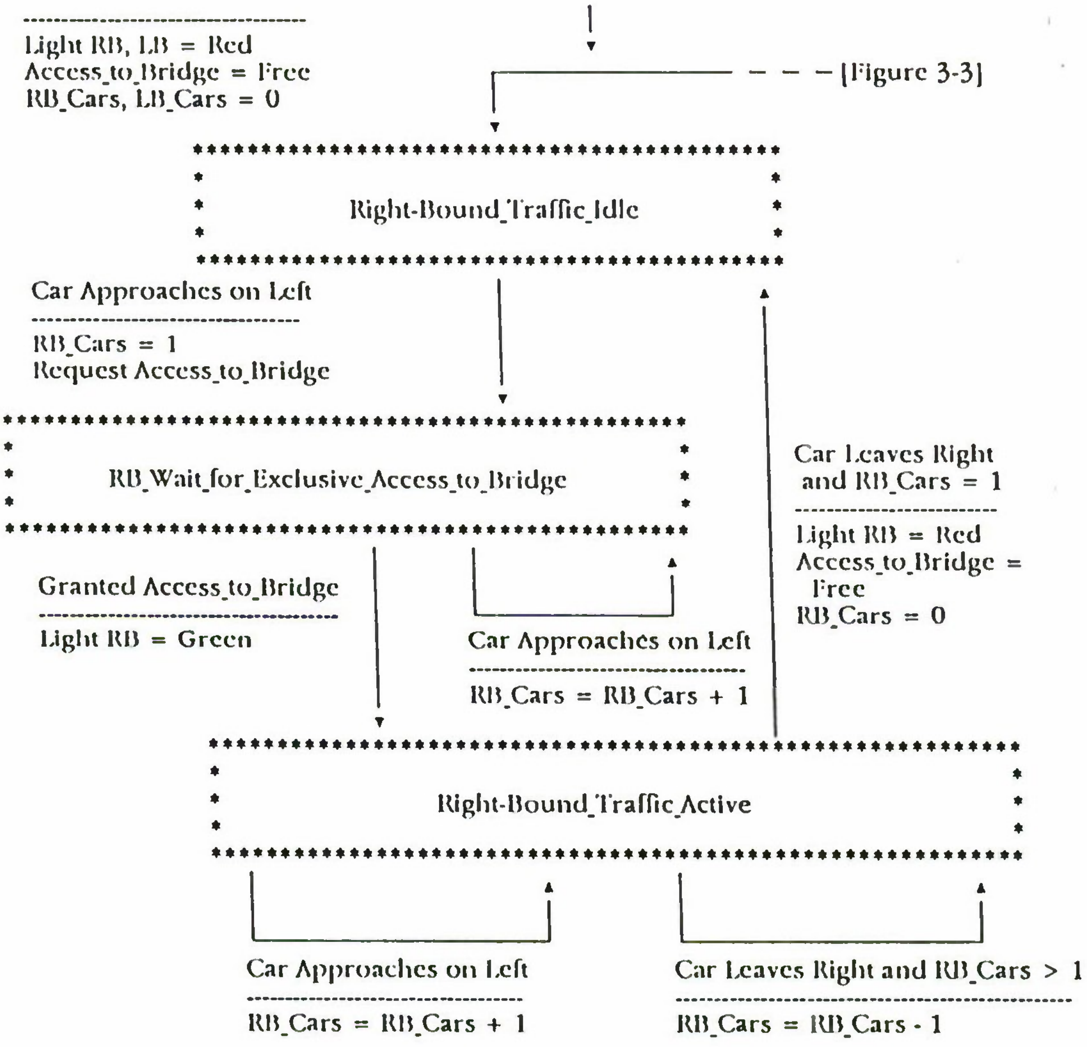

Light RB, LB = Red
Access_to_Bridge = Free
RB_Cars, LB_Cars = 0
[Figure 3-3]
Right-Bound_Traffic_Idle
Car Approaches on Left
RB_Cars = 1
Request Access_to_Bridge
RB_Wait_for_Exclusive_Access_to_Bridge
Car Leaves Right and RB_Cars = 1
Light RB = Red
Access_to_Bridge = Free
RB_Cars = 0
Granted Access_to_Bridge
Light RB = Green
Car Approaches on Left
RB_Cars = RB_Cars + 1
Right-Bound_Traffic_Active
Car Approaches on Left
RB_Cars = RB_Cars + 1
Car Leaves Right and RB_Cars > 1
RB_Cars = RB_Cars - 1

FIGURE 3-2: Graphical Representation of Behavioral Requirements: State Transition Diagrams

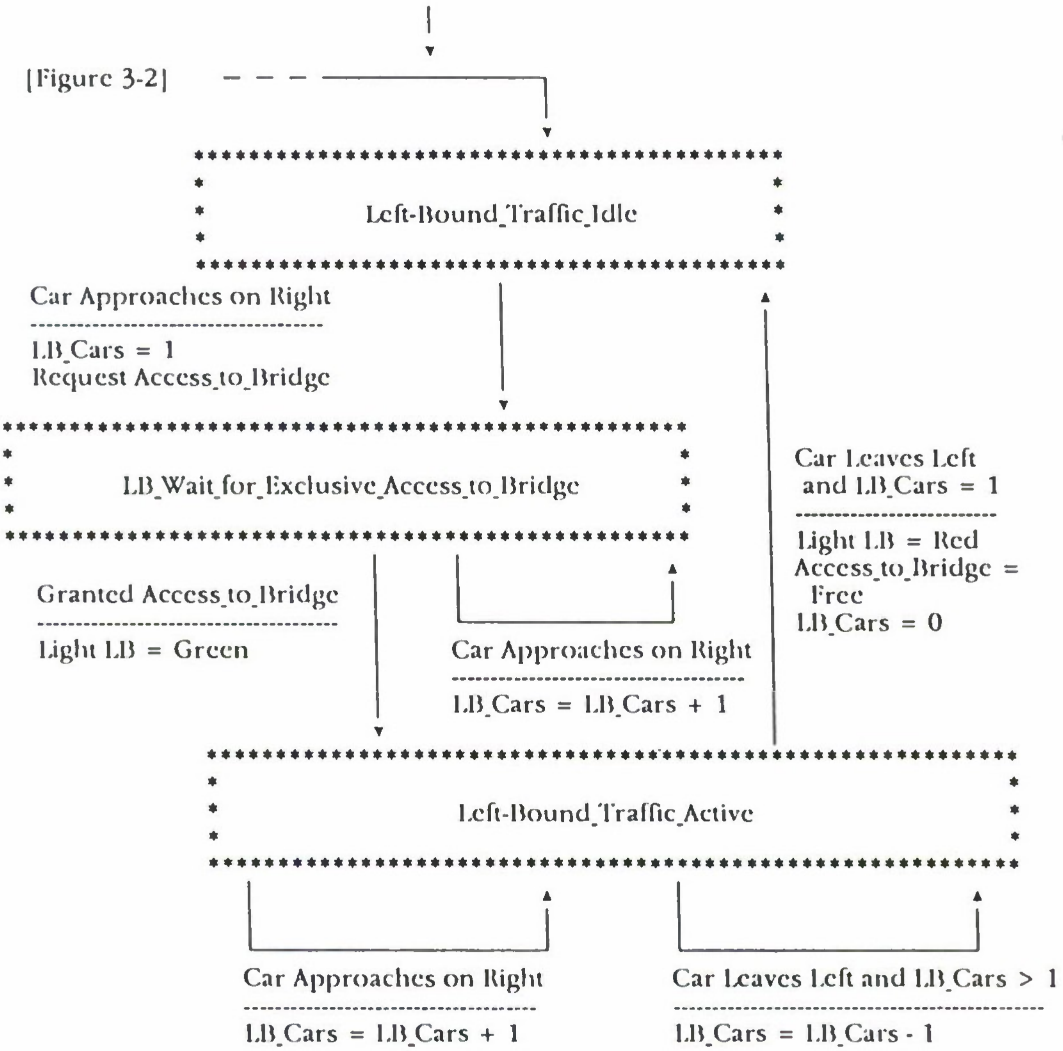

[Figure 3-2]
Left-Bound_Traffic_Idle
Car Approaches on Right
LB_Cars = 1
Request Access_to_Bridge
LB_Wait_for_Exclusive_Access_to_Bridge
Car Leaves Left
and LB_Cars = 1
Light LB = Red
Access_to_Bridge =
Free
LB_Cars = 0
Granted Access_to_Bridge
Light LB = Green
Car Approaches on Right
LB_Cars = LB_Cars + 1
Left-Bound_Traffic_Active
Car Approaches on Right
LB_Cars = LB_Cars + 1
Car Leaves Left and LB_Cars > 1
LB_Cars = LB_Cars - 1

FIGURE 3-3: Graphical Representation of Behavioral Requirements: State Transition Diagrams

LIST OF REFERENCES

1. Hammer, W., *Handbook of System and Product Safety*, Prentice Hall, Inc., 1972.

2. Bagchi, C., "Application of Fault Tree Analysis Techniques To Predict Instrumentation System Failure and Define Problem Areas in a Nuclear Power Plant Containment Study", *IEEE Transactions on Power Apparatus and Systems*, Vol. 11, pp. 4485-4492, November 1981.

3. Fussell, J. B., "A Formal Methodology for Fault Tree Construction", *Nuclear Science and Engineering*, Vol 52, No. 4, pp. 421-432, December 1983.

4. Salem, S.L., *A Computer-Oriented Approach to Fault Tree Construction*, Ph.D. Dissertation, University of California, Los Angeles, California, 1976.

5. Leveson, N. G., Harvey, P. R., "Analyzing Software Safety", *IEEE Transactions on Software Engineering*, vol.9, pp. 569-579, September 1983.

6. Neumann, P.G., "Letters From The Editor", *Software Engineering Notes*, Vol 8, No. 5, p. 3, October 1983.

7. Littlewood, B., "How to Measure Software Reliability and How Not To", *IEEE Transactions on Reliability*, vol. R-28,no. 2, pp. 103-110, June 1979.

8. Cha, S. S., Leveson, N. G., and Shimeall, T. J., "Fault Tree Analysis Applied to Ada", *Proceedings of the Tenth International Conference on Software Engineering*, Singapore, PP. 377-386, 1988.

9. Beizer, B., *Software System Testing and Quality Assurance*, p. 7, Van Nostrand Reinhold, 1984.

10. Ripps, D.L., *An Implementation Guide to Real-Time Programming*, Prentice Hall, Inc., 1990.

11. Department of Defense Military Standard MIL-STD-882B, *System Safety Program Requirements*, 28 June 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Computer Technology Programs, Code 37 Naval Postgraduate School Monterey, California 93943	1
4.	Department Chairman, Code CS Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
5.	Timothy Shimeall Code CS/Sm Computer Science Dept. Naval Postgraduate School Monterey, California 93943	5
6.	Nancy Leveson ICS Department University of California, Irvine Irvine, California 92717	1
7.	Robert E. Westbrook Code 31C Embedded Computer Technology Office Naval Weapons Center China Lake, California 92555-6001	1
8.	Stephen Cha ICS Department University of California, Irvine Irvine, California 92717	1
9.	J. R. Taylor Head of Institute Institute for Technical Systems Analysis Jernbanegade 52 A DK 400 Rosklide Denmark	1

10. Peter Neumann 1
SRI International
Menlo Park, California 94025

11. David Parnas 1
Computing and Information Science
Queen's University at Kingston
Kingston, Ontario, Canada K7L 3N6

12. Barry Daniels 1
Manager, Software Engineering: Systems
National Computing Centre Limited
Oxford Road
Manchester, England M1 7ed

13. George Dinsmore 1
TRW
1 Space Park 134/3816
Redondo Beach, California 90278

14. Wolfgang Ehrenberger 1
Fachhochschule
Fachbereich Informatik
6400 Fulda, West Germany

15. Bev Littlewood 1
Center for Software Reliability
City University
Northampton Square
London ECIV OHB, England

16. Donald Needham 2
7429 N. Oriole
Chicago, Illinois 60648

17. David Ripps 1
c/o Prentice Hall Publishers
Prentice Hall Building
Englewood Cliffs, New Jersey 07632

618-587